SEPIA
RT- ZBAS

Praktische Beispiele

Ein Real-Time Advanced BASIC Interpreter mit echtzeitfähiger Hardware-Unterstützung

Inhalt

Kapitel 1

Grundlagen
Einführung 3
Zahlengenauigkeit, Vorbelegung 4
Standard-Befehle (siehe auch Kapitel 2) 5
SEPIA RT-ZBAS Befehle 6
VGA-Farben, Text Color 7
Rechner starten, System booten 8

Erste Beispiele
Mein erstes Programm 10
CPU Temperatur erfassen, Temperatursensor 11
Quittungston mir XPBELL ausgeben 13
RTC Uhrzeit und Datum 14
RTC Programmbeispiel 15

Datenübertragung
Tastaturabfrage und Zeichenausgabe 16
XSTROUT, XSTRIN$ ser. I/O Datenübertragung 17
XGETKEY Zeichen über COM-Port lesen 18

Beispiele zur Hardware
LOADGO mit IDE-Karte und CF-Karte 19
MEM Memory-Karte ansteuern 20
WatchDog auf RTC-Karte programmieren 21
RL8 Relais programmieren 22
OI8 Optokoppler erfassen 23
AD1 analoge Spannungen messen 24
DA1 analoge Spannungen erzeugen 25
CNT Counter, 24 Bit Zähler auslesen 26
E32 32x Optokoppler erfassen 27
A32 32x Optokoppler programmieren 28

Spezielles
Die „AUTOBAS“ Startdatei 29
VGA Textfarben programmieren 30

IDE - Datenträger Befehle
DIR, LOAD, SAVE 31
DRIVE, RESET, RENAME,
ERASE, XROOT 32

Sonderfunktionen
16-Bit I/O-Adresse ansteuern 34 *
Debugger Funktionen 35
Hilfe zum Debugger, einzelne Menuefunktionen 37
IDE-Routinen im Debug Untermenue „S“ 38
ROM Adressen (Call) 40
VGA-Terminal Status melden 41
VGA-Grafik mit NET-Karte 42
EMail über XPort® senden 44
XPULSE PHI0/PHI90 Geber-Pattern erzeugen 45
XIPEAK Einzel-Impuls ab 5 µs erzeugen 47
XIRQON & XIRQOFF Interrupt-Generator 49

Anhang
Weitere Programmierbeispiele 50
ASCII - Tabelle 51
F-Tasten Belegung der PS/2-Tastatur 53
Datenblätter & Fotos (Web - Links) 54

Kapitel 2

Standard - Befehle
Einführung 56
Befehlsübersicht, implementierte Befehle 57

Fehlermeldungen
Software Fehlermeldungen 58
System Fehlermeldungen 58

Funktionen
Operatoren und math. Standardfunktionen 59

Variablen
Variablen und Variablen-Typen 60

Syntax

ABS 61
...
...
...
XOR 158

* Funktion zur Zeit noch in Erprobung

Einführung

In der Grundausstattung eines SEPIA-Systems werden neben BIOS, Debugger und Monitor, ein kl. OS-Betriebssystem, sowie ein textbasierender RT-Interpreter im EPROM mitgeliefert. Damit kann der Anwender SEPIA sofort starten und für seine Applikationen programmieren.

Folgende Punkte waren uns bei der Entwicklung ganz besonders wichtig:

- Größtmöglicher Anwendernutzen durch hohe Sicherheit und Vielseitigkeit
- Unabhängige Plattform mit eigenem BIOS und Betriebssystem (ohne weitere Kosten)
- Schneller Projekterfolg durch leichte Programmierung mittels einfacher Hochsprache
- Möglichkeit zum Debuggen und Einbringung von Binärcode in Maschinensprache
- Skalierbarkeit der Hardware durch vielfältige Schnittstellen (kein plug-and-pray)
- Nutzen von vorhandenen Ressourcen (bsp. VGA-Monitor, PS/2 Tastatur, CF-Card...)
- Möglichkeit zur Vernetzung über Ethernet (virtueller COM-Port für Windows-Systeme)
- Programmierung als Host-Rechner, oder fernbedienbar als Remote-I/O-System (RIO)

RT-ZBAS Advanced-BASIC ist eine leistungsfähige Programmiersprache, die einfach, intuitiv und sehr schnell lernbar ist. SEPIA kann somit fernab vom üblichen Marktgeschehen punkten, da Änderungen in der Programmierung, quasi vor Ort, jederzeit und ohne zusätzlichen PC durchführbar sind.

Die Wahl der Programmiersprache viel gleich zu Anfang auf ein echtzeitfähiges BASIC. Mit dieser einfachen Hochsprache können alle notwendigen Bedingungen für die Automation erfüllt werden. Somit schließt SEPIA die Lücke zwischen einfachen, kontaktplangesteuerten SPS-Steuerungen und hochkomplizierten, wie kostenaufwendigen, Automationssystemen. In der Praxis werden diese Systeme immer unübersichtlicher und dessen Handhabung bleibt oft nur noch wenigen Spezialisten vorbehalten, welche regelmäßig Fortbildungskurse bei den jeweiligen Herstellern belegen und teure Up-Dates nachbestellen und einpflegen müssen.

Für SEPIA benötigen Sie weder ein Seminar noch teure Up-Dates. Diese Dokumentation reicht vollkommen aus, da während der Entstehung alle Beispiele an praxisorientieren Aufbauten ausprobiert wurden. Diese lassen sich bei entsprechender Nachbildung sehr einfach wiederholen und nach eigenen Vorstellungen im Rahmen der angeboteten Möglichkeiten erweitern. Alle Beispiele wurden dazu an der Vollausstattung ausgetestet.

Die in dieser kleinen Dokumentation gezeigten Beispiele können, je nach Revisionsstand, in folgenden Angaben zu Ihrem Modell abweichen: BIOS-Version, Datum, Debugger/Monitor- und IDE-Version, Advanced-BASIC Version, RTC-Zeit und Datum, sowie Variablen wie RND und CPU-Temperatur. Die hier gezeigten Beispiele können nur dann genutzt werden, wenn die entsprechenden Erweiterungskarten im System physisch vorhanden sind. Die einzelne Komponentenadressierung orientiert sich dazu nach unserer MAPALL.ASM Empfehlung.

Symbole:
Alle wichtigen Hinweise sind mit diesem Kennzeichen versehen:

Merker und Hinweise zum Text werden mit einer roten Hand versehen:

Zahlengenauigkeit, vorbelegte Variablen, Strings und ASCII-Zeichen

Bei der Programmierung dürfen, ähnlich, wie bei einem Taschenrechner, bestimmte Grenzen und Bereiche nicht überschritten werden, da für alle Variablen nur ein bestimmter Speicherplatz zu Verfügung steht. Zudem werden bestimmte Variablen, wie die Kreiszahl „Pi“, mit festen Zahlen intern vorbelegt.

Hinweis: Das „**E**“ bedeutet x $^{10 \text{ hoch}}$

Bit	0/1 = Null oder Eins	
Character	0...127 (bsp. ASCII-Zeichen)	
Byte	00..FF, 0...255 Dez. oder 00000000...11111111 Bin.	
Ganze Zahlen, ohne Dezimalpunkt	-32768 bis +32767 einschl.	
Genauigkeit	-1.701411**E**+38 bis +1.701411**E**+38 einschl.	
PRINT EXP(88) <CR>	Anzeige: 1.651636255E+38	
Print EE <CR>	= Eulersche Zahl	Anzeige: 2.7182818285
Print Pi <CR>	= Kreiszahl	Anzeige: 3.1415926536
Zahlengenauigkeit	max. 12 Stellen	
Beispiel für Exponent 234.988E-7	= 0.0000235988	
Beispiel für Exponent 2359E6	= 2359000000	

int. Speicherplatzbedarf	
BASIC-Zeile	min. 5 Byte
Zeilennummer	min. 2 Byte
Pointer	min. 2 Byte
CR Zeilenvorschub	1 Byte
Stringvariable	min. 6 Byte
Aktive FOR-NEXT Schleife	16 Byte
GOSUB, ohne Return	6 Byte
String Bereich	max. 255 Zeichen pro String-Variable
BASIC-Zeilennummern	0 bis 65529
Länge einer Programmzeile	bis zu 255 Zeichen
Speicher	64 kB gesamt (ohne Erweiterung MEM-Karte)

Wichtig für Portzugriff, bsp. OUT, INP:		
Hexadezimale Zahlen, 8 bit	&00...&FF	entspricht 0...255
Hexadezimale Zahlen, 16 bit	&0000...&FFFF	entspricht 0...65535

Sprachen	
Ausgabemeldungen	deutsch
BASIC Programmierung	englisch
Unterstützte ASCII-Zeichen	US Zeichensatz ohne Umlaute (US/DE Tastatur)

Sonstiges	
VGA ASCII-Zeichenzatz	7/8 Bit, 0...127, ohne Umlaute
VGA Zeichen p.Sekunde	5000 bei 115.200 kBaud
Schleifenbefehl p. Sekunde	~ 4500 (0...100000 For-Next-Schleife bei 20 MHz)

In SEPIA verwendete Standard-Befehle

```
A: ABS, AND, ASC, AUTO, ATN
B: BYE
C: CHR, CLEAR, COS, CALL, CONT
D: DATA, DIM
E: EE, EDIT, ELSE, END, ERL, ERR, ERR1, ERROR, EQV, EXP, EXIT
F: FOR, FRE, FIX, FIX$, FIND
G: GOTO, GOSUB
H: HEX
I: INP, INPUT, IF, INKEY$, INSTR, INT, IMP
K: KILL
L: LET, LVAR, LIST, LOG, LOG10, LEN, LEFT$, LOCATE
M: MID$, MOD
N: NEW, NEXT, NOT
O: OUT, OUTBYTE, ON, OR
P: PI, POKE, POS, PEEK, PRINT, PRECISION, PRIVACY
R: READ, RESTORE, RETURN, RANDOMIZE, REM, RESUME, RENUMBER,
   RUN, RND, RND1, RIGHT$, REPLACE
S: STOP, SIN, SGN, SQR, STR$, STRING$, STEP, SPC
T: TAN, TRACE, TO, THEN, TAB
U: USING
V: VAL, VARPTR
W: WAIT
X: XOR
```

Eine ausführliche Erläuterung zu den hier verwendeten Befehlen finden Sie im Kapitel 2 sowie im HEBAS Benutzerhandbuch. Da HEBAS modular strukturiert wurde, sind in dieser Unterlage möglicherweise weitere Befehle erläutert, die in SEPIA jedoch aus Platzgründen nicht implementiert wurden. Dafür bietet SEPIA jedoch spezielle Advanced-Befehle für die Automation, auf welche wir in dieser Anleitung explizit eingehen werden. Im Anschluss finden Sie eine Übersicht zum erweiterten BASIC-Befehlssatz, welcher fortan Advanced-BASIC genannt wird. Manche Advanced-BASIC Beispiele können jedoch nur bei Verwendung entsprechender I/O-Hardware genutzt werden. So sind beispielsweise alle Laufwerks-Befehle wie DRIVE, DIR, LOAD, LOADGO und SAVE nur nutzbar, wenn SEPIA über ein IDE-Interface mit Datenträger (bsp. CF-Card) verfügt. Alle zeitlichen Angaben beziehen sich auf den Bus-Ausbau mit einer 20 MHz CPU-Karte + einem Waitstate.

Wichtiger Hinweis:
SEPIA prüft während des bootens nach, ob IDE-Interface und Datenträger vorhanden sind. Fällt die Prüfung negativ aus, wird eine entsprechende Mitteilung ausgegeben. So auch bei der RTC-Karte, da diese neben dem WatchDog eine Echtzeituhr enthält und bei fehlender Hardware keine Angaben zu Uhrzeit und Datum erzeugen kann. Die Nutzung der restlichen Hardware ist davon nicht betroffen, da im Remote I/O-Betrieb die IDE und RTC mit WDC möglicherweise nicht benötigt werden. Dies gilt gleichfalls für den VGA- und Ethernet-Port, da eine Remote-Verbindung direkt über den RS232-Port der CPU-Karte erfolgen kann.

SEPIA RT-ZBAS Advanced Befehle

VGA

```
XTCOLOR V,H              ; Textfarbe 0..9 setzen, V=Vordergrund, H=Hintergrund
XRSTCOL                  ; Textfarbe rücksetzen, org. Ausgabe Weiß auf Blau
XINVERT                  ; Textfarbe invertieren (rücksetzen mit XRSTCOL)
XSTSON                   ; Statusmeldung VGA-Terninal EIN
XSTSOFF                  ; Statusmeldung VGA-Terninal AUS
XGR.......               ; siehe Tabelle -> Sonderbefehle VGA Grafik
```

A/D und D/A Karte

```
XSETAD1(Port),Kanal      ; AD1-Karte Messkanal 1...16 auf MUX einstellen
x=XGETAD1(Port)          ; ließt von AD1 16 Bit Wandlung nach Variable x, oder
x=XRADC(Kanalx256+adr)   ; Kanal setzen und A/D lesen mit nur einem Befehl
XDAC(16Bit_Wert),Port    ; DA1-Karte dig. 12 Bit-Wert auf DAC-Port schreiben
XDASET,BasisPort         ; Latch + Freigabe für Output Switch (Parallelübergabe)
```

COM Port

```
XCOUT(x),byte            ; ein Byte auf COMx senden (x = 1,2,3,4,5,6,7)
byte = XCIN(x)           ; ein Byte von COMx empfangen (als Zahlenwert 0..255)
A$ = XSTRIN$(x)          ; empfängt eine Zeile mit 80 Zeichen von COMx
XSTROUT(COMx),A$         ; sendet einen String auf COMx (x = 1,2,3,4,5,6,7)
byte = XGETKEY(word)     ; ließt ein Zeichen von Tastatur auf COM1 nach Variable
```

I/O diverse

```
XHWRST                   ; Befehl ohne Parameter, löst Hardware SYSTEM-RESET aus
XIPEAK(byte),time,mode   ; erzeugt auf I/O-Adresse(byte) einen zeitlich def. Impuls
XPULSE(byte),word,mode   ; erzeugt auf I/O-Adresse(byte) bis 65535 Burst/Pattern
XGETEMP                  ; Text Ausgabe gibt CPU-Temperatur auf Konsole aus
XOUTWD(word),byte        ; auf 16Bit I/O-Adresse ein byte schreiben
byte = XINPWD(word)      ; auf 16Bit I/O-Adresse ein byte lesen
XWAITKEY                 ; Programm wartet endlos auf einen Tastendruck
XDELAY(word)             ; Delay 0...65535 (50000 = ca. 10 Sek.)
XPBELL byte,byte         ; Rechteck-Ton erzeugen: byte1: Tonhöhe, byte2: Tonlänge
XGETIME                  ; Text Ausgabe gibt Uhrzeit auf Konsole aus
XGEDATE                  ; Text Ausgabe gibt Datum auf Konsole aus
byte = XGETSEC(0)        ; von RTC0 aktuelle Sekunde nach Variable lesen
byte = XGETMIN(0)        ; von RTC0 aktuelle Minute nach Variable lesen
byte = XGETSTD(0)        ; von RTC0 aktuelle Stunde nach Variable lesen
XIRQON / XIRQOFF         ; Schaltet 1 Sek.Interrupt-Generator von RTC-Karte Ein/Aus
```

MEM - Befehle

```
XPOKE(Adresse),byte      ; Byte auf ext. MEM-Karte (hohe Adresse) schreiben
byte = XPEEK(Adresse)    ; Byte von ext. MEM-Karte (hohe Adresse) lesen
XSEC(x)                  ; 64k Segmentadresse vorwählen (x = 1...15)
```

„Disk" Direktbefehle auf IDE Interface mit CF/SD Card

```
XROOT x                  ; x = Verzeichnisnummer (0 ... max. 255)  0 = Root
RESET                    ; IDE-Medien neu initialisieren (IDE RESET-Command)
DIR                      ; Directory - Verzeichnis ausgeben
LOAD                     ; Programmdatei laden, gefolgt von Dateinummer
SAVE                     ; Programmdatei abspeichern, gefolgt von Dateinummer
RENAME                   ; Programmdatei umbenennen, gefolgt von Dateinummer
ERASE                    ; Programmdatei löschen, gefolgt von Dateinummer
LOADGO                   ; läd Programmdatei über Nummer und startet Anwendung
DRIVE 0                  ; auf Laufwerk 0 umschalten (IDE-Port0: Master)
DRIVE 1                  ; auf Laufwerk 1 umschalten (IDE-Port1: Slave)
DRIVE S                  ; Anschluss auf vorhandene Medien (Master & Slave) prüfen
DRIVE P                  ; Festplatte parken & in sleep-mode versetzen
DRIVE T                  ; Ausgabe Drive Status & Error Register, für Testzwecke
```

Fortsetzung

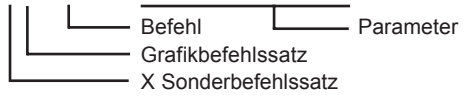
```
Farbtabelle zu Befehl: XTCOLOR V,H
je 10 VGA Textfarben für Vorder- und Hintergrundfarbe

          ; Farbe Vordergrund                ; Farbe Hintergrund
          ; 0 = Schwarz                      ; 0 = Schwarz
          ; 1 = Rot                          ; 1 = Rot
          ; 2 = Grün                         ; 2 = Dunkel-Grün
          ; 3 = Gelb                         ; 3 = Oliv/Khaki-Braun
          ; 4 = Blau                         ; 4 = Dunkel-Blau
          ; 5 = Rosa                         ; 5 = Mangenta
          ; 6 = Cyan                         ; 6 = Dunkel-Cyan
          ; 7 = Grau                         ; 7 = Hell-Grau
          ; 8 = Amber                        ; 8 = Hell-Blau
          ; 9 = Hell-Lila                    ; 9 = Orange
```

Grafik auf VGA-Monitor

Die Ausgabe von einfachen Diagrammen und Kurven erfolgt über einen gesonderten VGA-Mode mit 640 x 240 Pixel in zwei von 64 Farben. Damit die BASIC-Programmierung möglichst übersichtlich bleibt, wurden dazu folgende Advanced-Befehle etabliert:

```
XGRON                  ; schaltet VGA-Monitor in den Grafik-Mode
XGROFF                 ; schaltet VGA-Monitor wieder in den Text-Mode zurück
XGRESCOL               ; setzt die Farbpalette zurück (original)
XGRCLS                 ; löscht die gesamte Grafik
XGRSTCOL V,H           ; legt Vordergrund-Farbe und Hintergrund-Farbe fest
XGRTEST                ; erzeugt ein statisches Testbild mit Farbbalken
XGRPSET  X,Y           ; zeichnet einen Pixel an der Koordinate X,Y
XGRLINE  X1,Y1,X2,Y2   ; zeichnet eine Linie von X1,Y1 nach X2,Y2
```



Weitere Befehle und Beispiele:

```
A$ = INKEY$            ; ließt ein Zeichen von Tastatur (COM1) nach A$ ein
CLS                    ; Löscht Bildschirm + Cursor Pos. 1,1 (ANSI-BBS)
LOCATE x,y             ; Text-Cursor positionieren (ANSI-BBS)
BEEP                   ; kurzer Quittungston auf CPU-Karte ausgeben
RINGBELL               ; wie BEEP, jedoch mit ASCII-Code 07h auf COM1
var = INP(Port)        ; direkte Eingabe von I/O-Port (bsp. Optokoppler)
OUT(Port),byte         ; direkte Ausgabe auf I/O-Port (bsp. Relais)
var = INP(Port)        ; direkte Eingabe von I/O-Port lesen
OPTION                 ; Text Ausgaben formatieren
CTRL <E>               ; Tastenkombination für Programmabbruch
NEW                    ; BASIC-Programm im Speicher (RAM) löschen
BYE                    ; BASIC verlassen, zum Debugger-Menue wechseln
LIST                   ; BASIC-Listing anzeigen (Pause mit Leertaste)
RENUMBER <100,20>      ; Listing Zeilennummer neu formatieren
CONT                   ; nach Abbruch, Programm weiter ausführen
RUN                    ; ab Programmanfang starten
STOP                   ; Programm abbrechen
AUTO                   ; Automatische Zeilennummer bei Eingabe
ERROR <1..48>          ; Fehlermeldung von Nr. ausgeben
REPLACE                ; Programmtext ersetzen (zeilennummerorientiert)
TRACE                  ; Programmzeilen im Einzelschritt mitverfolgen
WAIT                   ; Ließt I/O-Port so lange, bis Bedingung erfüllt ist
```

Boot-Menue, EPROM-Inhalt, X-Sonderbefehle...

SEPIA RT-ZBAS Advanced-BASIC besitzt ein ca. 24 kB großes Standard-BASIC (inkl. mini BDOS), plus 8 kB erweiteres (Advanced) BASIC mit RT-Sonderfunktionen.

Ein Sonderbefehl kennzeichnet sich dadurch, dass dieser mit einem „X“ wie XPOKE beginnt und mit keinem „normalen“ BASIC vergleichbar, aber dennoch leicht verständlich ist.

Machen Sie dazu gleich den ersten Test und schalten Sie dazu SEPIA ein:

⊙ = Vorhanden
○ = nicht Vorhanden

- SEPIA booten (mit eingeschobener CF-Card auf IDE0 und IDE1)

```
** SEPIA PAC Industrie Computer **
-----------------------------------
BIOS 4.102A (C) H.Kolter & Dr. Hehl
20.01.2015

RAM DISK ..... 1 MB
MEM CP/M ..... ⊙
COM Init ..... ⊙
SIO Init ..... ⊙
PIO Init ..... ⊙
IDE Master ... ⊙
IDE Slave .... ⊙
IDE Init ..... ⊙

22:39:59  16.03.2015  Montag

CPU: +026 Grad C.

<ESC>    = Neustart
    C    = CP/M Boot >> MEM-Karte
    D    = Debugger
    L    = RAM loeschen
    K    = BASIC Kaltstart
    W    = BASIC Warmstart
    R    = REMOTE I/O
>_
```

Der Nutzer muss jetzt eine Eingabe tätigen: Taste D, L, K, W oder R. Im ersten Menue (s.o.) bitte zunächst die Taste „K“ für Kaltstart wählen/eingeben. Ausführliche Erläuterungen finden Sie dazu auf der nächsten Seite.

Taste „R“ versetzt das System unmittelbar in den Remote-I/O-Betrieb.
Taste „C“ gilt nur für speziellen CP/M-Betrieb in Verbindung mit MEM-Karte und Bootloader.

Booten von SEPIA nach einem RESET

Das System lässt sich grundsätzlich in zwei unterschiedlichen Betriebszuständen anwenden: Host- oder Remote-I/O-Mode. Im Host-Mode können zusätzliche Erweiterungen wie IDE, Ethernet, VGA, PS/2, MEM, sowie RTC mit WatchDog, oder weitere COM-Schnittstellen ergänzt werden. Im Remote-I/O-Betrieb sind diese Erweiterungen nicht zwingend notwendig, da alle Steuerbefehle von einem externen Rechner (i.d.R. PC) über eine COM-Schnittstelle erfolgen. Die Kommunikation der Steuerbefehle erfolgt in Klartext mittels Text-String$, so dass andere Laboranwendungen, oder aus Hochsprachen heraus, leichten Zugriff auf SEPIA erhalten. Alle Befehle sind dazu als Direkt-Befehl anwendbar. Für eigenständige Applikationen verwendet das System im Host-Betrieb, neben dem BIOS, ein kleines OS-Betriebssystem mit Debugger + Monitor für Maschinensprache, sowie ein textbasierter Advanced-BASIC-Interpreter für die Automation. Die Unterscheidung, ob Host- oder Remote-I/O-Betrieb, trifft der Anwender in der Startmenue-Abfrage. Im Host-Mode erkennt ein „Such-Modus“ automatisch, wenn eine CF-Karte (am IDE-Interface) eingeschoben ist, welche eine AUTOBAS Start-Datei enthält. So kann gleich nach dem Booten eine Initialisierung stattfinden, oder weitere Programme nachgeladen werden.

Nach einem Hardware-RESET startet das SEPIA-BIOS das Startmenue mit verschiedenen Eingabemöglichkeiten, die der Nutzer wählen kann. Während des bootens prüft das BIOS mehrere Schaltungen intern ab, um festzustellen, ob beipielsweise eine RTC oder IDE Karte verbaut wurde. Bei Fehlen dieser Hardware wird automatisch eine entsprechende Meldung auf die Konsole (COM1) generiert. Der Nutzer hat nun die Möglichkeit zwischen:

```
<ESC>    = Neustart
    D    = Debugger
    L    = RAM loeschen
    K    = BASIC Kaltstart
    W    = BASIC Warmstart
    R    = REMOTE I/O
```

auszuwählen, damit der weitere Verlauf seiner Programmierung stattfinden kann. Bei betätigen der ESC-Taste wird ein erneuter RESET ausgeführt (Warmstart), bei „D“ gelangt man direkt in den Debugger-Modus mit speziellen Befehlen, die eine maschinennahe Programmierung und Analyse zulässt. Die Taste „L“ löscht den gesamten SRAM-Speicher, so dass hier keine zufälligen Datenfragmente entstehen. „K“ startet den BASIC-Interpreter im Kalt-Modus, so dass alle internen BASIC-Variablen gelöscht und weitere Grundinitialisierungen vorgenommen werden. Mit der Taste „W“ werden diese Initialisierungen übergangen, so dass ein noch vorhandenes BASIC-Programm im SRAM-Speicher nutzbar bleibt. Mit dem „LIST“-Befehl kann man im BASIC-Modus dieses Progamm dann einsehen und bearbeiten. Somit besteht die u.a. Möglichkeit, auch nach einem Hardware-RESET (bei nicht unterbrochener Spannungsversorgung) ein BASIC-Prgramm im Speicher weiter nutzen zu können.

Die Taste „R“ startet den BASIC-Interpreter ohne Zeilennummer und Rückmeldungen, so dass hierbei eine einfache Remote-Kommunikation auf Basis der BASIC-Befehle über die COM Konsolenschnittstelle stattfinden kann. Da in diesem Modus ebenfalls alle BASIC-Befehle verwendet werden können, steht dem Nutzer eine Vielfalt von Einsatzmöglichkeiten, zur Ansteuerung der Hardware, zur Verfügung. Alle I/O-Leitungen können in dieser Betriebsart über „INP“ und „OUT“ oder den X... Advanced-Befehlen direkt angesteuert werden.

Mein erstes Programm...

Das BASIC meldet sich zunächst mit einer Abfrage zur Speichergrenze, da man beipielsweise später noch Maschinenprogramme in den oberen RAM-Speicher hinterlegen kann.

Drücken Sie zunächst nach „Speichergrenze“ die Return/Enter-Taste, bis folgende Meldung auf dem Monitor erscheint:

```
BASIC gestartet

Speichergrenze ? _

31821 Bytes frei

ADVANCED BASIC 6.02 (C) Dr.Hehl & H.Kolter

Fertig

_
```

Geben Sie nun folgenden Advanced-BASIC-Befehl mit abschließender Return/Enter-Taste ein:

```
XGETEMP
```

SEPIA meldet nun die aktuelle CPU-Temperatur (in diesem Fall +32 Grad Celsius)

```
+032 Grad C.
Fertig

_
```

Im nächsten Beispiel wollen wir nun die CPU-Temperatur wiederholt auf dem Bildschirm fromatiert ausgeben lassen, bis wir die Endlosschleife mit der Tastenkombination „CTRL+E“ manuell unterbrechen.

Weiter auf der nächsten Seite >>>

XGETEMP - Advanced Befehl

Da der BASIC-Programmspeicher noch leer ist, können wir gleich die ersten Programmzeilen eingeben. Löschen Sie aber zunächst den Bildschirm mit der folgenden Eingabe:

CLS gefolgt von der Return/Enter-Taste...

und geben Sie nach der „Fertig-Meldung“ folgende Programmzeilen ein:

```
Fertig

100 CLS
110 LOCATE 10,10
120 XGETEMP : PRINT
130 GOTO 110

RUN
```

Das Programm können Sie nun mit dem RUN-Befehl starten, gefolgt von der Return/Enter-Taste...

Die Monitorausgabe der CPU-Temperatur erfolgt jetzt permanent in der 10ten Zeile an der 10ten Spalte, bis eine manuelle Unterbrechung mit der Tastenkombination „CTRL+E“ (für End = Abbruch) erfolgt. Der Interpreter meldet den Abbruch immer mit Zeilennummer.

Durch ändern der X- und Y-Position im LOCATE-Befehl, lässt sich die Textausgabe auch an anderer Stelle ausgeben. Das in X-Richtung große Spaltenfeld hat max. 80 Zeichen, das in Y-Richtung 36 Zeilen. Größere Angaben sind nicht erlaubt. Die eingegebene X - und Y - Position zeigt immer auf das erste Zeichen einer Textausgabe.

Hinweis zur Hardware:
Der Temperatursensor befindet sich im 40 pol. DIL-Sockel unterhalb der CPU. Es handelt sich hierbei um einen TMP04 im TO92 Gehäuse der Fa. Analog Device. Der Sensor erzeugt dazu Rechteck-Impulse, die in einer Statemachine vom CPLD-IC in Echtzeit zu einer 32-Bit Zahl zeitlich vorberechnet wird. Die Differenz ergibt anschließend einen 16-Bit Wert, welcher proportional zur aktuellen Temperatur intern zur Verfügung steht. Damit die Temperatur in Grad Celsius angezeigt werden kann, muss die CPU (bzw. ein Z80 Unterprogramm) diesen Wert entsprechend umrechnen. Hierzu wird ein Divisor und ein Quotient benötigt, welche auch im BIOS-EPROM als Festwerte zur späteren Kalibration des Sensors hinterlegt sind.

Der Devisor liegt auf der EPROM-Speicherstelle 0020h, der Quotient auf 0023h.

Normales Beispiel zur Temperaturausgabe :

```
100 REM CPU-Temperatur ausgeben
110 CLS
120 LOCATE 10,5
130 PRINT „CPU-Temperatur = „; : XGETEMP
140 PRINT
150 XDELAY (10000)
160 GOTO 120
RUN
```

Hinweis zum Programm
Der Befehl XDELAY (arg. < 65535) verzögert den Programmablauf mit einer Variablen zwischen 0 und 65535. Während XDELAY ausgeführt wird, ist die Tastenkombination mit CTRL-E wirkungslos, da die Maschinenroutine in einer geschlossenen Schleife (DJNZ-Loop) ausgeführt wird, bis der Zählerstand erreicht ist. Die Variable wurde nur 16 Bit breit ausgelegt, damit keine unendlichen Verzögerungen programmiert werden können. Benötigt man dennoch längere Zeiten, kann der Befehl mehrfach hintereinander angewendet werden.

Bei einem 20 MHz-System liegt die XDELAY-Verzögerung bei dem Wert:

50000	bei etwa 10 Sekunden
5000	bei etwa 1 Sekunde
1000	bei etwa 200 ms
100	bei etwa 20 ms
10	bei etwa 2 ms

Ausführliches Beispiel: CPU-Temperatur über direkte I/O-Registerabfrage manuell mit BASIC berechnen

```
100 REM TMP04 Temperatursensor Messbereich -40...+120 Grad
110 REM Kalibrierte Temperaturkurve
120 :
140 OUT &18,8+4              : REM RESET internal two 16 bit counter
160 OUT &18,2+1              : REM enable counter + single shot meas
180 OUT &18,0                : REM get counts in 2 register from sensor
200 XDELAY(500)              : REM wait > 50 ms
220 :
240 A = INP(&18)             : REM INP-Reihenfolge nicht veraendern !!!
260 B = INP(&19)
270 C = INP(&1A)
280 IF C > 0 THEN GOTO 140   : REM bei ERROR ODER UEBERLAUF wiederholen
300 TY = A + (B*256)         : REM 16 bit Wert über zwei 8 bit Register bilden
320 TY = TY / 68             : REM Steilheit der Temperaturkurve (Divisor)
340 TY = TY - 192            : REM Grund-Offset zur Kalibration
350 :
360 PRINT USING „CPU Temp. = ###.#“, TY
380 GOTO 140
```

Quittungton ausgeben

Auf der CPU-Karte befindet sich ein kl. Piezo-Lautsprecher, um beispielsweise einen Quittungston zu erzeugen. Neben dem typischen BELL-Befehl, kann die Tonhöhe und Dauer durch folgenden Advanced-Befehl stattfinden: XPBELL 200,100.

In diesem Beispiel wurde die Tonhöhe 200 und Tondauer 100 vorgegeben. Die variablen Werte werden als Byte dezimal übergeben (0..255).

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Tonausgabe programmieren
110 XPBELL 50,250
120 XPBELL 80,250
130 XPBELL 100,250
140 XPBELL 120,250
150 XPBELL 140,250
160 XPBELL 160,250
170 XPBELL 180,250
180 XPBELL 200,250
190 XPBELL 255,255
```

Uhrzeit und Datum

Wenn Sie eine RTC-Karte verwenden, meldet sich SEPIA beim booten mit Zeit und Datum.

Die RTC-Karte beinhaltet, neben einer COM2 und LPT2 -Schnittstelle, einen Uhren-Chip und eine WDC WatchDog-Schaltung. Damit man den Uhren-Chip auch unter BASIC auslesen kann, wurden zwei Advanced-Befehle integriert: XGETIME und XGEDATE. Diese Befehle erzeugen eine einfache Print-Textausgabe der aktuellen Uhrzeit, oder dem Datum.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Uhrzeit & Datum unter BASIC ausgeben
110 PRINT "Zeit STD:MIN:SEK = "; : XGETIME
120 PRINT
130 :
140 PRINT "Datum TAG:MONAT:JAHR = "; : XGEDATE
150 PRINT
160 :
170 XDELAY(5000)
180 GOTO 110
```

Hinweis zum Programm
Der Befehl XDELAY(word) ermöglicht eine Ablaufpause.
GOTO 110 startet das Programm ab BASIC-Zeile 110 erneut.

Damit der Anwender die Uhrzeit über eine Variable mit in sein Programm einbinden kann, stehen andere Advanced-Befehle zur Verfügung: XGETSEC, XGETMIN und XGETSTD.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 CLS
110 A = XGETSEC(0)       : REM Liesst Sekunde von RTC 0
120 B = XGETMIN(0)       : REM Liesst Minute von RTC 0
130 C = XGETSTD(0)       : REM Liesst Stunde von RTC 0
140 :
150 LOCATE 10,10         : REM Cursorposition X 10, Y 10
160 :
170 PRINT "Uhrzeit = "; C; B; A; "     "
180 XDELAY(5000)
190 GOTO 110
```

Uhrzeit und Datum

Programmbeispiel, wenn die RTC-Register direkt ausgelesen werden sollen:

```
100 REM Uhr und Datum
120 REM RTC nur lesen
140 REM Setze HOLD, danach Read
150 :
160 OUT &3D,1      :REM HOLD
180 REM kann entfallen da BUSY immer da
200 WAIT &3D,2,0   : REM Maske "2" = 0010 = Bit 1
210 :
220 A = INP(&30)
230 B = INP(&31)
240 C = INP(&32)
250 D = INP(&33)
260 E = INP(&34)
270 F = INP(&35)
280 G = INP(&36)
290 H = INP(&37)
300 I = INP(&38)
310 J = INP(&39)
320 K = INP(&3A)
330 L = INP(&3B)
340 M = INP(&3C)
345 :
350 A1 = INP(&30) : IF A <> A1 THEN GOTO 210
360 B1 = INP(&31) : IF B <> B1 THEN GOTO 210
370 C1 = INP(&32) : IF C <> C1 THEN GOTO 210
380 D1 = INP(&33) : IF D <> D1 THEN GOTO 210
390 E1 = INP(&34) : IF E <> E1 THEN GOTO 210
400 F1 = INP(&35) : IF F <> F1 THEN GOTO 210
410 G1 = INP(&36) : IF G <> G1 THEN GOTO 210
420 H1 = INP(&37) : IF H <> H1 THEN GOTO 210
430 I1 = INP(&38) : IF I <> I1 THEN GOTO 210
440 J1 = INP(&39) : IF J <> J1 THEN GOTO 210
450 K1 = INP(&3A) : IF K <> K1 THEN GOTO 210
460 L1 = INP(&3B) : IF L <> L1 THEN GOTO 210
470 M1 = INP(&3C) : IF M <> M1 THEN GOTO 210
475 :
480 REM Loesche HOLD
500 OUT &3D,0      :REM HOLD
510 :
520 REM Dezimalsystem !!! deshalb nur 4 bit
540 SEK = A+(B*10)
560 MIN = C+(D*10)
580 STD = E+(F*10)
600 DAY = G+(H*10)
620 MO  = I+(J*10)
640 YER = 2000+K+(L*10)
660 WEK = M
670 :
680 PRINT "Datum:";DAY;".";MO;".";YER; "     Uhrzeit: ";STD;":";MIN;":";SEK,
700 If M = 0 then Print "Wochentag: Sonntag"
710 If M = 1 then Print "Wochentag: Montag"
720 If M = 2 then Print "Wochentag: Dienstag"
730 If M = 3 then Print "Wochentag: Mittwoch"
740 If M = 4 then Print "Wochentag: Donnerstag"
750 If M = 5 then Print "Wochentag: Freitag"
760 If M = 6 then Print "Wochentag: Samstag"
770 :
800 GOTO 100 : REM erneut auf Bildschirm ausgeben
```

Textausgabe und Tastatur/Zeichen-Eingabe

Sie können SEPIA im Host- oder Remote-Modus betreiben. Im Host-Betrieb (ähnlich einem PC) werden alle Ein/Ausgaben z.B. auf den VGA-Port der NET-Karte übertragen. Umgekehrt werden Tastatureingaben empfangen und der Konsole zur Verfügung gestellt. Im Remote-Betrieb werden diese Ein/Ausgaben über den COM1-Port der CPU-Karte behandelt. Die Zuweisung der Konsole erfolgt durch Festlegung per DIP-Schalter auf den E/A-Karten. Durch vertauschen der COM-Hardware-Adressen können somit Verbindungen zur Konsole, sowie über andere Karten, stattfinden. Die COM1 Schnittstelle ist dazu für die Konsole reserviert.

Um im Host-Betrieb (NET-Karte muss vorhanden sein) Tastatureingaben unter BASIC zu nutzen, kann man folgende Befehle anwenden:

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM INKEY$ und XWAITKEY TEST
110 CLS : LOCATE 2,2
120 PRINT "Eine Taste druecken..."
130 :
140 A$ = INKEY$ : IF A$ = "" THEN GOTO 140
150 :
160 PRINT "Die Taste war: "; A$
170 BEEP
180 PRINT "VARPTR-Zeiger: "; VARPTR(A$)
190 PRINT "Taste fuer weiter..."
200 XWAITKEY
210 GOTO 100
```

Der Advanced-Befehl „XWAITKEY“ unterbricht den Programmablauf und erwartet eine Tasteneingabe durch den User.

Eine andere Variante zum Vergleich:

```
100 REM Tastatur ASCII-Zeichencode anzeigen
110 CLS
120 LOCATE 2,2
130 B = ASC(INKEY$)
140 PRINT „ZEICHEN „; CHR$(B); „ DEZ.:“;B
150 IF B = 5 THEN STOP ELSE GOTO 130
```

Zeichensatz und Datenübertragung (1)

Das nachfolgende Programm zeigt, wie der Zeichensatz auf der Konsole (sprich auf COM1 über die ser. Schnittstelle, oder dem VGA-Bildschirm der NET-Karte) erzeugt werden kann.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Zeichensatz ausgeben
110 For X = 0 to 23
120 PRINT
130 NEXT X
150 PRINT CHR$(27);"[1J"
160 PRINT CHR$(27);"H"
170 PRINT CHR$(27);"[0,0H"
180 :
190 FOR Z = 30 TO 255
200 PRINT Z; CHR$(Z); "   ",
210 NEXT Z
```

Mit dem Advanced-BASIC-Befehl „XSTROUT(comnr.1..7)“ lassen sich ASCII-Zeichen als String zu einer anderen COM-Schnittstelle einfach umlenken bzw. ausgeben.

Beispiel:

```
100 DIM A$(10)          : REM Konsole -> Ethernet
110 A$ = „Hallo...“     : REM Zeichenkette nach A$
120 XSTROUT(6),A$       : REM String auf COM6 senden
```

Es geht auch umgekehrt:

```
100 DIM A$(80)          : REM Ethernet -> Konsole
110 A$ = XSTRIN$(6)     : REM lese String auf COM6
120 Print A$            : REM nach <CR> String auf Konsole
```

Achtung kein „G“ wie String, sondern $. Bitte Syntax beachten

Oder zur Ausgabe ein einzelnes Zeichen auf COM1...7:

```
100 A = 66              : REM Character ASCII-Zeichen „B“
110 XCOUT(7),A          : REM Zeichen auf COM7 ausgeben
```

Zeichensatz und Datenübertragung (2)

Möchte man die Schnittstelle extern überprüfen, kann man beispielsweise Daten als Echo auf dem gleichen COM-Port zurücksenden.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Daten von COM2 empfangen und als Echo zuruecksenden
120 Z = XCIN(2)            : REM Zeichen von COM2 empfangen
130 IF Z > 0 THEN PRINT CHR$(Z);   : REM auf Konsole
140 XCOUT(2),Z             : REM auf COM2 zurück senden
150 IF Z = 13 THEN PRINT   : REM LF bei CR mit ausgeben
160 GOTO 120               : REM Loop Endlosschleife
```

Möchte man ein Zeichen direkt von der Tastatur einlesen, ohne in einer Endlosschleife zu verharren (bsp. INKEY$), kann der Befehl „var = XGETKEY(timeout)“ angewendet werden. Das ASCII-Zeichen wird mit 8-bit dezimal an die Variable „var“ übergeben. Der Timeout-Wert kann dazu von 0 bis 65535 eingestellt werden. Wird der Timeout zu klein gewählt, kann das zu lesende Zeichen verpasst werden, ist der Wert zu groß eingestellt, sind lange Wartezeiten bei der Eingabe an der Tastatur die Folge. Ist der Empfangspuffer des UART leer, erfolgt die Ausgabe mit dem Wert Null. Nach Ablauf von Timeout (Faktor ca. 10 x µs) kann das BASIC-Programm anschließend fortgeführt werden.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 A = XGETKEY(20000)        : REM lese Zeichen nach A
110 IF A > 0 THEN PRINT CHR$(A);
120 IF A = 13 THEN PRINT      : REM auf CR folgt LF
125 :
130 IF A > 0 THEN XCOUT(2),A : REM auf COM 2 mitschreiben
135 :
140 GOTO 100
```

In Zeile 110 wird das Zeichen auf COM1 (Konsole) ausgegeben. In Zeile 130 wird das gleiche Zeichen auf COM 2 gesendet. Somit lässt sich mit ein paar wenigen BASIC-Befehlen ein einfaches Terminal-Programm erstellen.

LOADGO - Befehl

Bei Verwendung der IDE-Karte mit einem Datenspeicher (CF oder SD Card) kann mit dem Direktbefehl „LOADGO(x)“ aus einem laufenden BASIC-Programm ein anderes Programm geladen und gestartet werden. Somit lassen sich beispielsweise auch mehrere Programme über eine eigene Menuestruktur verwalten, oder ggf. größere Programme in mehrere, kleine Unterprogramme zerlegen, die dann je nach Bedarf nachgeladen werden. Da alle BASIC-Programme über Programmnummern verwaltet werden, wird die Zieldatei als Nummer zugewiesen. Unabhängig davon, darf der Dateiname bis zu 32 ASCII-Zeichen beinhalten. Die Dateinummer erfolgt hierbei in hexadezimaler Schreibweise (&00 ... &FF).

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Beispiel für LOADGO(&xx)
110 CLS
120 LOCATE 2,3
130 PRINT "Menue mit Unterprogrammen"
140 PRINT "-------------------------"
150 PRINT
160 PRINT "1 - MASTER-SLAVE         "
170 PRINT "2 - ZIDE6                "
180 PRINT "3 - RELAIS TEST          "
190 PRINT "4 - TONERZEUGUNG         "
200 PRINT
210 :
220 INPUT W$
230 :
240 ON VAL(W$) GOTO 260,270,280,290
250 :
260 LOADGO(&06) : REM Starte Dateinummer
270 LOADGO(&04) : REM Starte Dateinummer
280 LOADGO(&15) : REM Starte Dateinummer
290 LOADGO(&17) : REM Starte Dateinummer
```

Anmerkung:
Die Dateien &06, &04, &15 und &17 müssen auf dem Datenträger vorhanden sein und jeweils ein startfähiges BASIC-Programm beinhalten.

Datei-Verzeichnis ausgeben:
Nutzen Sie den „DIR“-Befehl, um das gesamte Directory des Datenträgers anzuzeigen.

MEM - Segmentadressierung und externe Speicherverwaltung

Damit das SEPIA-BASIC auf den erweiterten Speicher zugreifen kann, wurde zusätzlich eine kleine MMU in das CPLD integriert. Sie ermöglicht eine Segmentadressierung mit 16k-Pages zu je vier Blöcken pro 64k Segment. Die Speicherverwaltung wird über die I/O-Adresse 0000h gesteuert. Hier wurden nun neben den Adressleitungen A16...A19 und BANKEN ein PG0-Bit und PG1-Bit für das Mapping erweitert, welche vier 16k-Pages je nach Bedarf umschaltet. Der Grund für diese Maßnahme war, dass man größere Datenmengen (z.B. Messdaten) nicht in das eigene BASIC-RAM ablegt, sondern in einen Schattenspeicher der genug Platz bietet.

Damit der Anwender sich nicht um die Umschaltung der einzelnen Pages kümmern muss, wurden im Advanced-BASIC weitere Befehle integriert: „XPOKE“ und „XPEEK“. Hiermit lassen sich direkt 64k Schreib- und Leseoperationen je Segment komfortabel handhaben. Die Segementumschaltung erfolgt zuvor über den Befehl „XSEG 1...15“.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 CLS : LOCATE 2,2
110 PRINT „SEPIA SRAM & MEM-CHECK 32/512/1MB „
120 PRINT „================================ „
130 PRINT
140 POKE (&F000),&A5
150 B = PEEK(&F000)
160 IF B = (&A5) THEN PRINT „32 KB Programmspeicher o.k.“
170 PRINT
200 :
210 FOR X = 1 TO 15
220 XSEG(X)
230 XPOKE (&F000),&A5
240 B = XPEEK(&F000)
250 IF B = (&A5) THEN PRINT „Test 64k Segment „;X
260 NEXT X
```

Anmerkung:
Um die Funktionen der Speichererweiterung nutzen zu können, benötigen Sie die MEM-Karte mit mindestens einem SRAM Speicher-Chip.

Der Befehl „XSEG“ gibt die jeweilige Segmentadresse (64k Block) vor. Die absolute RAM-Adressierung muss gemäß der tatsächlichen Speicherbestückung entsprechen.

WDC Watchdog auf RTC-Karte programmieren

Die gesamte WDC-Programmierung erfolgt über drei I/O-Register: OUT &2C,x (0..15) setzt 16 verschiedene Zeitfenster für den Time-out, OUT &2D,1 erzeugt die Freigabe der WDC-Funktion, und mit OUT &2F,0 wird ein Re-Trigger programmiert, welcher vor Ablauf des Zeitfensters erfolgen muss, damit kein RESET ausgelöst wird. Das Relais schaltet mit einer Haltedauer von ca. 1 Sekunde (bei 8 MHz CPU-Takt) und löscht dabei das Time-out-Flag im CPLD, sodass keine Rückkopplung mit der RESET-Logik entsteht. Die RESET-Schaltung der CPU-Karte übernimmt anschließend alle Aufgaben der Neuinitialisierung (Warmstart). Der WDC-Befehl OUT &2D,0 schaltet die Funktion manuell ab und setzt das Time-Out-Flag wieder auf Null (disable), so dass man gezielt Routinen mit - oder - ohne WDC-Unterstützung programmieren kann.

Bei einer gewollten Unterbrechung durch den Anwender (Tastenkombination „CTRL-E“) wird über das BIOS das WDC-Flag ebenfalls rückgesetzt, da sonst bei jeder Umprogrammierung des BASIC-Quellcodes ein unnötiger CPU-RESET durchgeführt würde.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM WDC-TEST (Watch-Dog)
110 OUT (&2C),6   : REM Timeout Zeitbasis waehlen 0..15
120 OUT (&2D),1   : REM 1=Enable 0=Disable WDC
130 :
140 OUT (&2F),0   : REM Re-Trigger vor Timeout
150 :
160 REM Hier steht eingekapseltes Hauptprogramm
170 XDELAY(2000)  : REM simuliert die Programmdauer
180 :
190 GOTO 140      : REM WDC-Schleife
```

Anmerkung zur Hardware:
Abhängig vom Master-Clock wird ein Frequenzteiler mittels Vergleicher im CPLD intern vorprogrammiert, um verschiedene Time-Out Zeitfenster zu bestimmen. Ein weiteres Kontroll-Register schaltet die WatchDog-Funktion ein bzw. aus. Danach muss auf einem I/O-Port ein regelmäßiger Zugriff durch das Hauptprogramm erfolgen. Unterbleibt der Zugriff innerhalb des programmierten Zeitfensters (bsp. Softwarefehler), schaltet das Relais den RESET-Vorgang ein, damit der Rechner automatisch neu starten kann. Dabei schaltet das Relais mittels Ruhestrom als Öffner (Fail-Safe). Der anschließende RESET-Vorgang setzt die Watch-Dog-Funktion wieder zurück in eine neutrale Ausgangsposition (Rückfallebene), bis erneut eine gewollte Freigabe durch das Programm erfolgt. Der Relaiskontakt kann zudem für eine Weiterschaltung (bsp. als Folgerelais, Alarmkontakt, Meldelinie..) genutzt werden. Die RESET-Leitung (zur CPU) kann zusätzlich durch einen Jumper je nach Bedarf unterbrochen werden, so dass der Relaisausgang für andere Zwecke nutzbar ist.

Relais auf RL8-Karte programmieren

Alle Relais lassen sich auf dieser Erweiterungskarte direkt byte-weise ansteuern, so dass ein OUT-Befehl auf der zugehörigen Kartenadresse die Kontaktstellung entsprechend abändert.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM RELAIS-TEST mit XDELAY (Lauflicht)
110 FOR I = 0 TO 7    : REM Relais 1..8 in Schleife testen
120 POW2 = 2 ^ I
130 :
140 OUT &B0,INT(POW2) : REM Relais gesetzt
150 PRINT „SET RELAIS Nr.:";I+1
160 :
170 XDELAY(500)   : REM 100ms Verzoegern (bei 20MHz CPU)
180 B = INP(&B0)  : REM Kontakt-Zustand zuruecklesen
190 :
200 IF INT(POW2) <> B THEN PRINT „Fehler bei Relais: „;B
210 NEXT I
220 XDELAY(1000)
230 OUT &B0,0     : REM Alle Relais AUS
```

Hinweise zur Hardware:
Die 8-fach Relais-Baugruppe verfügt über 2 Amp. Doppelkontakte. Pro Relais ist ein UM-Kontakt über einen 25 pol. Sub-D Stecker an der Frontplatte zugängig. Der zweite Relais-Kontakt wird der Schaltung, zur Erhöhung der Eigensicherheit, wieder rückgeführt, sodass der Schließer-Zustand, per Software, Bit-weise auf der gleichen I/O-Adresse von jedem Relais wieder zurückgelesen werden kann. Zur zusätzlichen Kontolle dient eine I/O-LED, rechts neben der Power-LED, die bei jedem OUT- Befehl eingeschaltet, sowie bei jedem IN-Befehl wieder ausgeschaltet wird. Damit lassen sich I/O-Zugriffe auf das Relais-Register optisch mitverfolgen. Die Adresslage der Karte ist variabel einstellbar. Dazu wird ein Byte aus 255 möglichen I/O-Adressen für den Zugriff mit DIP-Schalter (DIP-SW1) ausgewählt.

RESET, WatchDog, Rückfallebene:
Ein DIP-Schalter (DIP-SW2) entscheidet mit Schalter Nr.1, ob der System-RESET die Relais-Ausgabe löschen soll, oder der aktuelle Registerinhalt unabhängig vom RESET beibehalten werden soll. Dies kann bei manchen Anwendungen sehr wichtig sein, die nicht durch einen RESET verändert, oder unterbrochen werden dürfen. Die gleiche Möglichkeit besteht ebenfalls, bei einem Time-Out vom WDC (WatchDog) den 8-Bit Registerinhalt zu löschen und damit die Relais in eine ursprünglich stromlose Ausgangsposition zu versetzen. Da der Rechner, selbst nach einem kompletten RESET, den Relais-Zustand der Karte wieder einlesen kann, sind sehr sichere Applikationen realisierbar die sonst nur teuren High-End-Systemen vorbehalten waren.

Optokoppler auf OI8-Karte abfragen

Alle Optokoppler lassen sich auf dieser Erweiterungskarte direkt byte-weise abfragen, so dass ein INP-Befehl auf der zugehörigen Kartenadresse das Eingangs-Register lesen kann.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
110 O = INP(&B8)      : REM Register-Eingang lesen
120 :
130 PRINT „OPTO-INPUT HEX = „; HEX$(O)
140 :
150 FOR I = 8 TO 0 STEP -1
160 POW2 = 2 ^ I      : REM Bit (Opto-Kanal) maskieren
165 :
170 IF INT(O AND POW2)=INT(POW2) THEN Bin$=Bin$+"1" ELSE
Bin$=Bin$+"0"
175 :
180 NEXT I
190 PRINT „OPTO-INPUT BIN = „; Bin$
```

Hinweise zur Hardware:
Alle Eingänge werden wie bei der RL8 komfortabel über LEDs angezeigt. Dazu werden die Signale im CPLD intern zwischengespeichert und anschließend erst zur LED-Anzeige gebracht, damit eine sichere, optische Gegenkontolle der erfassten Signalzustände möglich ist. Ein interner Vergleicher prüft alle eingehenden High- Pegel mit der Anzeige auf Übereinstimmung. Bei Zustandsänderung blinkt die zugehörige Kanal-LED so lange, bis der Rechner mit einem Lesebefehl den Zwischenspeicher übernommen hat. Der Lesevorgang wird durch eine weitere LED angezeigt. Somit ist eine zusätzliche Kontolle zur Übergabe des Registers an die Rechner- CPU gegeben. Der hohe Aufwand an Kontollmöglichkeiten bietet dem Anwender erhebliche Vorteile bei der Programmierung, da jedes einzelne Bit, ob high oder low, während der Erfassung mitverfolgt werden kann.

Analoge Messwerte mit AD1 erfassen

Mit der AD1-Karte lassen sich analoge Spannungen zwischen +/- 10 Volt oder kleineren Messbereichen komfortabel erfassen. Zur Messung sind nur zwei Befehle nötig:

XSETAD1(Port),Kanal ... stellt den analogen Messkanal ein
D = XGETAD1(Port) ... gibt das Messergebnis als 16 Bit-Wert an eine Variable zurück

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Messwert auf Kanal 1 von A/D-Karte einlesen
110 REM Basisadresse 90 Hex, Messbereich +/- 10 Volt
120 :
130 XSETAD1(&91),1    : REM A/D Messkanal einstellen
140 :
150 D = XGETAD1(&90)  : REM Wandlung 16BIT Digits nach D
160 :
170 V = (D * 0.0003051804379) - 10.000 : REM Messwert
180 PRINT „Volt = „;  : PRINT USING"##.####"; V
```

Hinweise zur Hardware:
Die AD1-Messkarte dient zur Erfassung analoger Messwerte mit 16-Bit Auflösung. Die A/D-Karte wird in mehreren Genauigkeitsklassen und Geschwindigkeiten angeboten. Je nach Bestückung kann die Messkarte Spannung oder Strom (0...20mA) messen. Dazu sind entsprechende Strom-Shunts vorgesehen, welche auch als Spannungsteiler beschaltet werden können. Als high-speed, low-power 16-bit A/D converter dient ein Wandler vom Typ AD976 (Hersteller: Analog-Device®).

Diese analoge Messkarte benötigt für die gesamte Programmierung nur zwei Bytes im I/O-Adressraum. Um hohe Abtastraten zu erzielen, wurde das Advanced-BASIC mit entsprechenden X-Befehlen erweitert. Die R/C (read-convert) Steuerung, EOC (end-of-conversion) Abfrage, sowie low- und high-byte Bildung werden bereits in Maschinensprache intern vorverarbeitet, so dass ein fertiger 16 Bit Wert an eine Variable übergeben werden kann.

Speziell für den Remote-Betrieb gibt es noch eine kürzere Variante als BASIC-Befehl:

```
100 REM Messwert auf Kanal 5 von A/D-Karte einlesen
110 K = 5
140 D = XRADC(K*256+(&90)) : REM direkte Wandlung nach D
150 :
160 V = (D * 0.0003051804379) - 10.000 : REM Messwert
170 PRINT „Volt = „;  : PRINT USING"##.####"; V
```

Analoge Spannung mit DA1 erzeugen

Mit der DA1-Karte lassen sich 4 analoge Spannungen zwischen +/- 10 Volt oder kleineren Bereichen komfortabel erzeugen. Zur Spannungsausgabe sind nur wenige X-Befehle nötig:

XDAC(Word),Adresse..... setzt den digitalen Spannungswert 0..4095 auf die DAC-Adresse
XDASET(Kartenadresse)... übergibt die DAC-Werte in den Latch und gibt Ausgang frei

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Bipolar-Mode alle DAC-Kanäle 0 Volt:
120 XDAC(&800),&98    : REM setze Wert auf Kanal 1
130 XDASET(&98)       : REM DAC Latch & Output Freigabe
140 XDAC(&800),&9A    : REM setze Wert auf Kanal 2
150 XDASET(&98)       : REM DAC Latch & Output Freigabe
160 XDAC(&800),&9C    : REM setze Wert auf Kanal 3
170 XDASET(&98)       : REM DAC Latch & Output Freigabe
180 XDAC(&800),&9E    : REM setze Wert auf Kanal 4
190 XDASET(&98)       : REM DAC Latch & Output Freigabe

Beispiel 2:
100 REM Kanal 1 und Kanal 2 gleichzeitig ausgeben:
110 XDAC(&0A00),&98   : REM setze +2,5 Volt auf Kanal 1
120 XDAC(&0FFF),&9A   : REM setze +10 Volt auf Kanal 2
130 XDASET(&98)       : REM DAC Latch & Output Freigabe
```

Hinweise zur Hardware:

Die DA1 Karte bietet je nach Bestückung wahlweise zwei oder vier D/A-Kanäle mit je 12 Bit Auflösung an. Je nach Referenz-Chip (REF01 / REF02) und Jumper-Einstellung können Ausgangsspannungen von +/- 10 Volt, 0...10 Volt, oder +/- 5 Volt, 0...5 Volt erzeugt werden. Die Wandler arbeiten intern mit Latch-Zwischenspeicher, so dass alle Kanäle auch zeitgleich übergeben werden können. (siehe Beispiel 2)

Die Grundeinstellung uni. / bipolar erfolgt für jeden Wandler-Baustein getrennt. Da bei dieser Karte Dual-ICs verwendet werden (zwei Wandler in einem DIP-Gehäuse), gilt die Einstellung immer paarweise. Die I/O-Basisadresse ist einstellbar. Dazu wird die Adresslage für den Port-Zugriff mit DIP-Schalter DIP-SW1 ausgewählt.

Insgesamt benötigt die Karte 8 Register. Zwei Lese-Register dienen zur Übergabe der neuen DAC-Werte, sowie einem schaltbaren Ausgang, welcher während der RESET-Phase abgeschaltet bleibt (disable), um keine ungewollten Spannungszustände zu erzeugen. Alle I/O-Kartenzugriffe werden durch eine LED zusätzlich angezeigt. Werkseitig verfügt das Interface über eine 9pol. D-Sub Buchse. Weitere Angaben können Sie dem Datenblatt entnehmen.

Zählerstände (digitale counts) mit CNT-Karte erfassen

Mit der CNT-Karte lassen sich digitale TTL-Impulse komfortabel zählen, oder Positionen an inkrementalen Gebern (bsp. Winkelgeber, Längenmeßstab, Messtaster...) erfassen.
Zur Messung sind nur wenige Input-Befehle nötig.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Zaehler 1 auf Basisadresse A0 auslesen
110 A = INP(&A0) : REM Low, Read speichert alle Register
120 B = INP(&A1) : REM Medium
130 C = INP(&A2) : REM High
140 D = INP(&A3) : REM nur Vorzeichen-Bit in Data Bit 0
150 :
160 X = A + (B*256) + (C*65535) : REM Zählerstand berechnen
170 PRINT „Alle Register: „
180 PRINT A;B;C;D
190 PRINT „Counter 1 = „;X
```

Hinweise zur Hardware:
Die CNT-Messkarte dient der Erfassung von dig. Impulsen. Beide Zähler können Pulsraten von 0 bis 10 MHz in Echtzeit verarbeiten. Die Impulserfassung erfolgt intern über schnelle Quadraturencoder. Damit keine Impulse ausbleiben werden 3x 8-Bit Latch plus ein Sign-Bit, in einem vierten Register, immer zwischengespeichert. Die Zähler arbeiten vollkommen unabhängig, können jedoch über einen Triggereingang ebenso synchron betrieben werden. Dadurch können beliebig viele Karten parallel betrieben werden, um beispielsweise die Ist-Position bei Längenmessungen in Mehrachsensystemen ohne Zeitjitter zu erfassen. Die anschließende Verarbeitung wird mit absoluten Zählerständen vorgenommen.

Auf Grund der int. 20 MHz Zeitbasis ist eine Impulserfassung (up/down-count) bis 10 MHz möglich. Im 4-fach Inkremental-Mode beträgt die Abtrastrate 5 MHz. Diese Einstellung wird beispielsweise für digitale Drehgeber und Encoder (mit diff. TTL/CMOS-Signal) oder inkrementale Längenmeßstäbe an Mehrachsen-Systeme und CNC-Maschinen benötigt. Ein Zähler-Reset kann synchron oder asynchron erfolgen, sowie über die internen Counter als Zählerstand vorprogrammiert werden. Je Zähler werden vier I/O-Register verwendet. Die Zwischenspeicherung erfolgt bei Lesen des Low-Bytes. Während der Datenübertragung der Registerinhalte, zählt der Counter im Hintergrund weiter, damit kein Verlust an der augenblicklichen Position des Gebers entsteht. Über einen DIP-Schalter werden feste Parameter zu jedem Counter separat eingestellt. Die I/O-Basisadresse ist einstellbar. Dazu wird die Adresslage für den Port-Zugriff mit DIP-Schalter DIP-SW1 ausgewählt.

Optokoppler auf E32-Karte abfragen

Alle Optokoppler-Eingänge lassen sich auf dieser Erweiterungskarte direkt byte-weise erfassen, so dass INP-Befehle auf der zugehörigen Kartenadresse das jeweilige Eingangs-Register einlesen kann.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Opto-Kanal 1..32 einlesen (vier 8-Bit Register)
120 CLS : LOCATE 2,4
140 :
150 A = INP(&B8)
160 B = INP(&B9)
170 C = INP(&BA)
180 D = INP(&BB)
190 :
200 PRINT „1. BYTE INP = „;A
210 PRINT „2. BYTE INP = „;B
220 PRINT „3. BYTE INP = „;C
230 PRINT „4. BYTE INP = „;D
```

Hinweis zur Hardware:
Da die 32-bit Karte insgesamt 4 Register zu je 8-bit verwendet, erfolgt die Abfrage auf vier unterschiedlichen I/O-Adressen.

Optokoppler-Ausgänge auf A32-Karte programmieren

Alle Optokoppler-Ausgänge lassen sich auf dieser Erweiterungskarte direkt byte-weise erzeugen, so dass OUT-Befehle auf der zugehörigen Kartenadresse das jeweilige Ausgabe-Register überschreiben kann. Das byte wird hierzu invertiert übergeben, damit bei einem Hardware-RESET alle Ausgänge (lachtch-up bedingt) hochohmig (=high-Z) sind.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Ausgabe Opto-Kanal (1. Register)
110 X = 128           : REM setzte Bit 7 (0x80h)
120 OUT (&BC), 255-X  : REM Output
```

Hinweis2 zur Hardware:
Da die 32-bit Karte insgesamt 4 Register zu je 8-bit verwendet, erfolgt die Ausgabe auf vier unterschiedlichen I/O-Adressen. Jedes 8-bit-Register dient gleichzeitig als Zwischenspeicher, so dass erst durch „überschreiben“ des Bytes ein neuer Wert für das jeweilige Register auf den Optokoppler-Ausgang übertragen wird. Es stehen drei verschiedene Kartenvarianten zur Wahl: P- oder N-schaltend, sowie Photo- MOS Relais bis 350 mA pro Kanal. Es sind auch Mischbestückungen mit P, N und Photo-MOS möglich. Durch Verwendung einer invertierten Logik mit Freigabe-FF (erster Kartenzugriff), werden keine undefinierten Pegelzustände während der Power-Up- und RESET-Phase erzeugt. Ein WatchDog-Reset versetzt alle Ausgänge in eine neutrale Ausgangssituation, wie man sie nach dem Einschalten des Rechners vorfindet (alle Ausgänge = OFF bzw. hochohmig, high-Z).

Die „AUTOBAS“ Startdatei im Hauptverzeichnis (Root)

Eine Besonderheit kommt der ersten Datei auf dem Datenträger (Drive0) zu, denn sie kann, ähnlich einer .bat Batch-Datei, ein BASIC-Programm direkt nach einem RESET ausführen, ohne weitere Benutzereingaben zu tätigen. Dazu muss das Programm zwingend unter dem Namen **AUTOBAS** in **Großbuchstaben** als Datei **01** abgespeichert sein.

Achtung:
SEPIA prüft während des bootens, ob die IDE-Karte nebst Speichermedium (IDE0) und diese Datei vorhanden sind. Falls ja, wird das Programm unmittelbar vom System geladen und im BASIC-Interpreter gestartet. Das AUTOBAS BASIC-Programm ermöglicht somit ein Wiederanlauf von individuellen Programmen, beispielsweise nach einem Stromausfall.

Mit LOADGO können ebenso weitere Programme aus AUTOBAS heraus gestartet werden, sowie COM-Schnittstellen und/oder I/O-Ports neu initialisiert werden. Bei der Nutzung von AUTOBAS gilt die Verwendung des WDC (WatchDog) zu berücksichtigen. Auch hier wird bei einem RESET die AUTOBAS-Datei automatisch geladen und gestartet. Da in AUTOBAS der gesamte Befehlsvorrat des BASIC-Interpreters verwendet werden kann, sind ebenso Protokollausgaben, z.B. auf einem Drucker möglich, welche dann Zeit, Datum, sowie andere Angaben zu diesem Ereignis schriftlich festhalten. Wird keine AUTOBAS Datei im Root-Verzeichnis abgespeichert, startet SEPIA das ursprüngliche Menue zur weiteren Abfrage.

Bildschirm-Befehle bei Verwendung der NET-Karte (mit VGA-Ausgang)

Auf den ersten Seiten wurden VGA-Farb-Befehle nur kurz vorgestellt. Die Handhabung und Wirkungsweise dieser Advanced-BASIC-Befehle sollen nun näher erläutert werden. Bei Textausgaben auf ext. Terminals oder PC-Terminal-Software, kann die Farbausgabe je nach Gerät unterschiedlich ausfallen. Intern erfolgen die verwendeten Steuerkommandos bzw. ESC-Sequenzen in Anlehnung an ein VT100 Terminal.

Beispiel: **`XTCOLOR(0,9)`**
Die gesamte Textausgabe erfolgt mit schwarzen Zeichen auf hell-blauem Hintergrund.

Beispiel: **`XTCOLOR(8,0)`**
Die gesamte Textausgabe erfolgt mit amber Zeichen auf schwarzem Hintergrund.

Beispiel: **`XRSTCOL`**
Der Befehl enthält keine Zusatzparameter. Die anschließende Textausgabe erfolgt in der boot-Einstellung. (Weisser Text auf dunkelblauen Hintergrund)

Beispiel: **`XINVERT`**
Der Befehl enthält keine Zusatzparameter. Die anschließende Textausgabe (ab dem Cursor) erfolgt farblich invertiert, bis die Invertierung wieder aufgehoben wird.

```
; Farbe Vordergrund
; 0 = Schwarz
; 1 = Rot
; 2 = Grün
; 3 = Gelb
; 4 = Blau
; 5 = Rosa
; 6 = Cyan
; 7 = Grau
; 8 = Amber
; 9 = Hell-Lila

; Farbe Hintergrund
; 0 = Schwarz
; 1 = Rot
; 2 = Dunkel-Grün
; 3 = Khaki-Braun
; 4 = Dunkel-Blau
; 5 = Mangenta
; 6 = Dunkel-Cyan
; 7 = Hell-Grau
; 8 = Hell-Blau
; 9 = Orange
```

HDD-DISK-Befehle bei Verwendung der IDE-Karte (1)

Auf den ersten Seiten wurden IDE (Disk)-Befehle nur sehr kurz vorgestellt. Die Handhabung und Wirkungsweise dieser Advanced-BASIC-Befehle sollen nun näher erläutert werden.

Beispiel: **DRIVE 0**

Durch Eingabe des Befehls wird der interne Laufwerkzeiger auf Laufwerk 0 (Master) voreingestellt. Alle nachfolgenden Operationen wie DIR, SAVE, LOAD, ERASE... beziehen sich fortan auf dieses Laufwerk. In der Regel handelt es sich dabei um die CF-Karte an IDE0.

Beispiel: **DRIVE 1**

Durch Eingabe des Befehls wird der interne Laufwerkzeiger auf Laufwerk 1 (Slave) voreingestellt. Alle nachfolgenden Operationen wie DIR, SAVE, LOAD, ERASE... beziehen sich fortan auf dieses Laufwerk. In der Regel handelt es sich dabei um die SD-Karte an IDE1 oder ein SSD bzw. Flash-Drive.

Beispiel: **DRIVE S**

Durch Eingabe des Befehls werden alle aktuellen (vorhandene) Laufwerke angezeigt.

Beispiel: **DRIVE P**

Durch Eingabe des Befehls wird das aktuelle Laufwerk geparkt. (Köpfe auf Spur 0)

Beispiel: **DRIVE T**

Durch Eingabe des Befehls werden Register-, Fehler-, und Statusmeldungen zum aktuellen Laufwerk angezeigt.

Beispiel: **RESET**

Durch Eingabe des Befehls werden alle HDD-Register rückgesetzt bzw. grundinitialisiert.

Beispiel: **DIR**

Durch Eingabe des Befehls wird das Disk-Inhaltsverzeichnis angezeigt. Es kann je Laufwerk und Verzeichnisebene bis zu 255 Dateien enthalten. Nach 16 Einträgen pausiert die DIR-Ausgabe. Sie kann durch betätigen der Leertaste fortgeführt oder durch Eingabe J/N auch abgebrochen werden. Die Einträge werden nach Dateinummer 0..FF sortiert ausgegeben.

Beispiel: **ERASE**

Durch Eingabe des Befehls wird nach der Dateinummer gefragt, welche anschließend nach J/N-Abfrage/Bestätigung gelöscht werden soll. Die nun freie Dateinummer kann anschließend mit einem neuen BASIC-Program überschrieben werden, oder frei bleiben.

HDD-DISK-Befehle bei Verwendung der IDE-Karte (2)

Fortsetzung:

Beispiel: **LOAD**

Durch Eingabe des Befehls wird zunächst nach einer Dateinummer (0..FF) gefragt. Das Programm mit der angegebenen Dateinummer wird nach Bestätigung in den RAM-Seicher geladen. Ein erfolgreicher „Load“ wird nach Übertragung durch einen kurzen Quittungston signalisiert. Anschließend kann man mit „List“ das Programm sichten, ggf. drucken, nachberabeiten oder mit „RUN“ starten.

Beispiel: **RENAME**

Durch Eingabe des Befehls wird zunächst die Dateinummer eingegeben, welche einen neuen Dateinamen erhalten soll. Im zweiten Abschnitt wird der Dateiname überschrieben und abgespeichert. Alle Eingaben sind dialog-geführt. Der Dateiname kann alle ASCII-Zeichen (32...127 dez.) enthalten und ist bis zu 32 Zeichen lang.

Beispiel: **XROOT(x)**

Durch Eingabe des Befehls wird der Zeiger für die Verzeichnisebene vorbelegt mit der anschließend alle Dateioperationen durchgeführt werden. Abhängig vom Speichermedium (HDD Kapazität) können bis zu 255 Verzeichnisse verwaltet werden. Die Ebenen-Nummer entspricht dem Cylinder-High aus der ATA CHS-Tabelle. Besitzt ein Medium lediglich 15 Cylinder-High-Einträge (siehe CHS-Tabelle), können folglich nur 15 Verzeichnisse genutzt werden. Bei voller Unterstützung mit 255 low und 255 high Cylinder können je Medium bis zu 65280 Dateien verwaltet werden. Es können jedoch nur Dateinummern von 1..255 vergeben werden, da das erste Root-Verzeichnis (0) wichtige Daten für das Directory enthält. Somit ist dieser Cylinder für den Nutzer i.d.R. tabu. Eine Sicherheitsabfrage verhindert zusätzlich den unerlaubten Zugriff auf diesen Bereich.

Beispiel: **LOADGO(x)**

Durch Eingabe des Befehls kann über die Dateinummer (x) ein anderes BASIC-Programm nachgeladen und sofort gestartet werden. Somit lassen sich beispielsweise auch mehrere Programme untereineander verschachteln. Die Dateinummer erfolgt in sedezimaler Schreib-weise (&00 ... &FF).

HDD-DISK-Befehle bei Verwendung der IDE-Karte (3)

Fortsetzung:

Beispiel für eine Directory-Ausgabe auf dem Bildschirm. Die DIR-Ausgabe pausiert nach 16 Einträgen. Durch Tastendruck (bsp. „J“ oder Leertaste) wird die Ausgabe weiter gescrollt, oder mit „N“ abgebrochen.

```
Drive: 0  akt. Verzeichnis: 000
 Datei: Nr., Name, Zeit, Datum, Groesse

 01 AUTO........................ 23:32:52  02.06.2012  3628 Bytes
 02 TEST ....................... 11:22:59  25.12.2011  0072 Bytes
 03 CRT-TEST.................... 12:14:29  25.12.2011  0103 Bytes
 04 Tonleiter.BAS............... 20:40:30  27.12.2011  0175 Bytes
 05 TONLEITER.BAS............... 20:46:03  27.12.2011  0175 Bytes
 06 AUTOBAS-BEEP................ 20:54:55  24.05.2012  0072 Bytes
 07 AUTOBAS..................... 23:32:43  02.06.2012  3628 Bytes
 08 16BIT-R/W I/O............... 21:12:25  03.06.2012  0194 Bytes
 09 STEP........................ 13:33:58  03.06.2012  0071 Bytes
 10 EXIT........................ 13:53:18  03.06.2012  0073 Bytes
 11 XPOKE-XPEEK................. 21:42:19  03.06.2012  0283 Bytes
 12 FIX$........................ 20:54:44  03.06.2012  0073 Bytes
 13 XSCALL...................... 21:41:27  03.06.2012  0217 Bytes
 14 OUTBYTE..................... 14:58:52  04.06.2012  0052 Bytes
 FF TEST AUTOBAS................ 12:26:51  24.12.2011  0072 Bytes


 weiter ? Taste!

 BASIC verfuegbar



 Fertig
```

Kurze Erläuterung
Die Anzeige „Drive: 0“ bedeutet, das hier das Laufwerk0 an IDE-0 ausgewählt wurde.
Die Anzeige „Verzeichnisebene: 000“ bedeutet, das die Directory-Ausgaben dem ersten Verzeichnis (root) zuzuordnen sind. Je nach Medium (siehe CHS-Tabelle: Cylinder-High) können bis zu 255 Verzeichnisebenen angelegt (und sogar einzeln formatiert) werden.
Die Formatierung erfolgt über das Debugger-Menue mit „S“ zum SEPIA IDE-Menue mit dem Tastenkürzel „F“. Jede Datei kann zudem beliebig oft mit neuen Programmen überspielt, oder einzeln gelöscht werden.

16 Bit I/O-Adressen direkt ansteuern

Mit SEPIAs Advanced-BASIC können 16-Bit Adressen auf dem Adressbus generiert werden. Dazu wird das BC-Register des Prozessors verwendet, indem die Adressen A8...A15 auf „B“ und A0...A7 auf das „C“-Register eingeschrieben werden. Werden anschließend INP- oder OUT-Befehle verwendet, liegt die Adresse mit dem IORQ-Signal an, sodass eine Dekoderschaltung diese als 16-Bit-Adresse interpretieren kann. Zur Unterscheidung wurden hierzu weitere Advanced-Befehle integriert, die diese Zugriffe behandelt:

```
XOUTWD(word),byte       ; auf 16Bit I/O-Adresse ein byte schreiben
byte = XINPWD(word)     ; auf 16Bit I/O-Adresse ein byte lesen
```

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM 16 Bit I/O Test
120 CLS
140 LOCATE2,2
160 XOUTWD(&B0),55          :REM Schreibe 55 nach B0h (Relaiskarte)
180 XDELAY(1000)
200 X = XINPWD(&B0)         :REM Lese den Port zurück, nach X
220 PRINT "Register = ";X
240 :
260 XDELAY(2000)
280 XOUTWD(&FFB0),1         :REM Überschneidung mit B0h !!!
300 XDELAY(2000)
320 Y = XINPWD(&FFB0)       :REM Lese den Port zurück, nach Y
340 PRINT "Register = ";Y
```

ACHTUNG:
Verwenden Sie diese Befehle sehr vorsichtig. Erweiterungskarten besitzen i.d.R. keine volle Adressdekodierung, so dass es bei falscher Programmierung zu Überschneidungen mit Datenkollision kommen kann. Die Anwendung dieser X-Befehle ist erst für Adressen ab E0h vorgesehen, da hier im Normalfall keine Erweiterungskarten mehr installiert sind.

Zu beachten gilt weiterhin, dass der SEPIA-Systembus im full-speed-mode mit 20 MHz betrieben wird. Langsame Port-I/O Bausteine können in diesem Fall nur mittels Wait-State-Zyklen an diese hohe Bus-Geschwindigkeit manuell angepasst werden. Dazu bietet der SEPIA-Bus-Contoller (ispLSI1032) mittels DIP-Schalter mehrere Einstellmöglichkeiten zur IORQ-Verzögerung > siehe techn. Doku: SEPIA-Bus.

Sonderfunktionen: Debugger < > BASIC

Es besteht die Möglichkeit zwischen Debugger- und BASIC-Mode nach belieben hin-und-her zu schalten. Benötigt man beispielsweise den Debugger zur Einsicht von aktuellen RAM Speicherdaten, so kann der BASIC-Mode durch den Befehl „**Bye**“ verlassen werden, ohne das flüchtige Daten oder Zeiger zerstört werden. Im umgekehrten Fall kann der Debugger-Betrieb mit „W“ wie Warmstart verlassen werden, um sich wieder dem BASIC-Programm zu widmen. Wird der Debugger mit „K“ verlassen, werden alle Zeiger und Teile von Variablen wie Initialisierungstabellen neu gesetzt = überschrieben wie RESET-Zustand.

Die Besonderheit dieser Vorgehensweise erlaubt dem Nutzer die optionale Einbringung von Z80 Maschinencode in den BASIC-Speicher**. Durch CALL(Adresse) wird dieser implizierte Code unter BASIC ansprechbar. Bitte nicht vergessen: Sprungadresse am Schluss mit RET beenden. Wird dieses BASIC-Programm anschließend auf einen Datenträger abgespeichert, so wird auch der implizierte Code mit übernommen. Dies hat den Vorteil, das man BASIC und Z80 Maschinensprache miteinander beliebig mischen, speichern und zurückladen kann.

**
In diesem Fall ist es ratsam, sich vorher einen Teilbereich des BASIC-Speichers für die Größe seines Codes zu reservieren, damit keine Überschneidungen von Speicherbereichen stattfinden können. Die Reservierung kann gleich zu Beginn (beim Start) des BASIC-Interpreters erfolgen. Die Return-Taste nimmt keine Speicherreservierung vor und weist den gesamten RAM-Speicher dem Interpreter zu (i.d.R. = 31821 Byte).

Rechenbeispiel:
Das SRAM besitzt urspünglich 32768 byte Speicherzellen. Davon werden i.d.R. 947 Byte für interne Zwecke wie Zwischenparameter und Variablen: Stack-Buffer, Drive-Tabelle, String-Buffer, Uhrzeit- und Sektoren-Buffer verwendet. Um eine genauere Speicherbelegung der einzelnen Bufferbereiche zu ermitteln, kann im Debugger die * Taste weiter Auskunft über die gängigen Labels geben. Darin enthalten sind ebenso BASIC-Anfang und BASIC-Ende-Adresse, sowie aktuelle Stapeladresse und freier RAM-Bereich.

Debugger - Menue

Der Debugger umfasst folgende Befehle:

```
DEBUGGER V2.41

 22:04:59  03.06.2012  Sonntag

 S = IDE Editor
 L = RAM loeschen & Neustart
 D = RAM anzeigen ab Adresse
 I = Befehle interpretieren ab Adresse
 F = RAM fuellen, von, bis, Wert
 A = ASCII-Text eingeben (max. 256)
 G = Starten ab Adresse, Breakpoint
 T = Einzelschritt ab Adresse
 C = Einzelschritt (ohne CALL)
 E = RAM aendern ab Adresse
 Y = Wert1, Wert3, ...suchen
 R = DEBUG-Register, Wert
 X = DEBUG-Register loeschen
 P = Port I/O: I77 oder O77,35
 U = Uhrzeit/Datum stellen
 M = CPU-Temperatur
 H = Protokoll-Drucker: EIN/AUS
 N = XSEG Segmentregister
 * = Label Adressen
 Z = Zurueck
 <Esc> = Neustart, K = Kaltstart, W = Warmstart

 >
```

Das SEPIA-IDE-Menu kann direkt mit der Taste „S“ angewählt werden. Hier trifft der Anwender weitere Möglichkeiten zur Diagnose, oder kann beispielsweise einzelne Sektoren überschreiben bzw. formatieren (siehe nächste Seite).

Alle Debugger- und Monitor-Funktionen sind weitgehend selbsterklärend gestaltet, um einen einfachen und schnellen Einstieg in die maschinennahe Programmierung zu bieten. Die Schreibweise erfolgt i.d.R. Hexadezimal, die Ausgabe (als Hex-Dump) kann mit Z80-Labels (quasi wie ein Disassembler) aus dem RAM-Speicher direkt zurück interpretiert werden.

Wer einen separaten Crossassembler für den PC besitzt, kann Z80 Maschinenprogramme extern linken und den Binärcode beipielsweise in ein EPROM übertragen. Da die Speicher- und Port-Schnittstellen alle in der MAPALL.ASM dokumentiert sind, kann der Programmierer hier sein eigenes System, unabhängig von SEPIA, etablieren.

Debugger - Menue

DEBUGGER Hilfe im Einzelnen

S = Springe zum IDE Editor-Menue

L = SRAM löschen und Neustart (booten)

D = Zeige Speicher-Dump: D 100 <CR> (Adresse hex), D 8000

I = Zeige Z80-Codes: I 100<CR> (Adresse hex), I 8000
(Anzeige wie ein kleiner Disassembler)

F = Fuelle RAM von,bis,Wert: F 9000 9200 45 <CR>

A = ASCII-Eingabe: CNTRL-D beendet

G = Starte ab Adresse, Breakpoint: G 9000,9003 <CR>'

T = Step ab Adresse: T 9000 <CR>

C = Step (ueberspringe CALL) ab Adresse: C 9000 <CR>

E = RAM aendern ab Adresse: E 7000 <CR>
Leertaste = vorwaerts
Minustaste = zurueck
<CR> Ende

Y = Wert1, Wert2, usw. suchen: Y FE 1A <CR> Alle Register auf 0 setzen: X <CR>

R = Register zeigen: R <CR>, RB0130 <CR> Register BC Wert 0130,
R Register Wert: z.B. BC loeschen: RB0 <CR> A=AF, B=BC, D=DE, H=HL, X=IX, Y=IY, P=PC, S=SP'
Stapel Wert setzen: RS8000 <CR>: , RZB0130 <CR> Zweitregister BC Wert

P + I = Input PORT I/O Adresse. Direkteingabe: PI7F,
Ausgabe in Hex und Binär. Leertaste beendet Port I/O

P + O = Output PORT I/O Adresse,Wert. Direkteingabe: PO34,FF <CR>
<CR> wiederholt Befehl
Leertaste beendet Port I/O Ausgabe

Z = Zurück, zum Debugger-Menue (erneuter Aufruf dieser Textseite)

K = Kaltstart (RAM wird gelöscht, alle Schnittstellen werden neu initialisiert)

W = Warmstart (alle RAM-Variablen bleiben erhalten)

<ESC> Neustart

SEPIA IDE - Menue

Das IDE-Menue umfasst folgende Befehle:

```
IDE Editor

 V = DRIVE Master(0) Slave(1)
 I = Identify
 M = MBR lesen
 E = Error/Status zeigen
 X = Drive INIT
 Y = IDE-RESET
 Q = 1 Sektor lesen, ablegen, anzeigen
 R = Sektoren lesen (max. 3Eh)
 T = Sektoren schreiben
 O = Sektoren ueberschreiben
 D = Directory
 J = Verzeichnis Nr.?
 F = Formatieren J/N
 N = Dateiname aendern
 L = Datei laden
 P = Programm starten
 S = Datei speichern
 H = Hardcopy EIN/AUS <CR>
 Z = Zurueck -> Debugger

 <Esc> = Neustart, K = Kaltstart, W = Warmstart
 >
```

Das Menue ist weitgehend selbsterklärend: Mit „V“ 0 oder 1 wird der IDE-Datenträger Master oder Slave aktiviert, mit „I“ wird die Identifikation des Datenträgers als Hex- und ASCII-Dump angezeigt, mit „M“ lässt sich der Master-Boot-Record auslesen. Mit Taste „E“ wird der IDE-Controller angewiesen, das jeweilige Medium auf Fehler zu prüfen. Mit „X“ wird die Drive-Tabelle neu übertragen, mit „Y“ wird das Medium neu gestartet. Mit Taste „Q“ kann man einen bestimmten Sektor direkt vom Medium laden (512 Byte), ins RAM ablegen und anschließend anzeigen lassen. Mit „R“ und „T“ werden Sektoren direkt gelesen/geschrieben oder mit „O“ überschrieben. Die Taste „D“ hat die gleiche Funktion wie der DIR-Befehl in BASIC. Mit „J“ kann die Verzeichnisebene (Zylinder High bis max. 255) gewechselt werden um weitere Dateien zu verwalten. Mit „F“ wird der Datenträger auf dem jeweiligen Zylinder formatiert. Somit kann man einzelne Verzeichnisse formatieren, ohne Daten in anderen Verzeichnissen zu zerstören. Mit „N“ kann ein Dateiname über die Dateinummer wieder neu vergeben werden. „L“ funktioniert wie LOAD unter BASIC und dient zum laden von gespeicherten Dateien in den SEPIA-RAM-Speicher. Mit „P“ lässt sich ein Programm direkt starten, mit „S“ wird eine Datei mit Dateinummer und Name abgespeichert. Alle Text - Ein/Ausgaben lassen sich mit „H“ auf den Drucker mit protokollieren. „Z“ beendet das Menue und kehrt zum Debugger zurück.

SEPIA IDE - Menue: Taste „I“

```
>I = Identify

 Drive: 0
 RAM-Puffer-Adresse:

       .0..1..2..3..4..5..6..7..8..9..A..B..C..D..E..F 0....5....A....F

 8170  03 32 FD EA 00 00 00 22 00 00 00 2C 04 4A 0F 20 .2.j..."...,.J..
 8180  00 00 00 10 7E 00 02 00 00 3F 00 3B 8E 00 00 00 .........?.;....
 8190  37 41 41 46 30 37 31 33 30 45 36 41 30 30 30 30 7AAF07130E6A0000
 81A0  30 35 33 38 00 02 00 02 00 04 56 65 72 36 2E 30 0538......Ver6.0
 81B0  34 20 43 46 20 43 61 72 64 20 20 20 20 20 20 20 4.CF.Card.......
 81C0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ................
 81D0  20 20 20 20 20 20 20 20 20 20 80 01 00 00 2B 00 ..............+.
 81E0  40 00 02 00 00 00 00 07 0F 20 00 10 00 3F 8E 00 @............?..
 81F0  00 3B 01 01 8E 00 00 3B 00 00 04 07 00 03 00 78 .;.....;.......x
 8200  00 78 00 78 00 78 40 00 00 00 00 00 00 00 00 00 .x.x.x@.........
 8210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 30 00 ..............0..
 8220  70 2B 54 04 40 00 70 00 14 04 40 00 00 7F 00 1E p+T.@.p...@.....
 8230  00 1E 00 00 FF FE 00 00 00 00 00 00 00 00 00 00 ................
 8240  00 00 00 00 8E 00 00 3B 00 00 00 00 00 00 00 08 .......;........
 8250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
 8260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
 8270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................


 weiter ? Taste!
```

Interessante ROM - CALLs

Es besteht die Möglichkeit, mittels CALL-Befehl, ROM-Routinen direkt aus dem BASIC heraus zu starten. In der folgenden Tabelle sind einige nützliche Sprungadressen aufgelistet, die dem Programmierer, der Erfahrung mit Maschinensprache hat, Arbeit abnehmen können.

Die Eingabe hinter dem CALL-Befehl erfolgt in Hex als 8 oder 16 Bit Adresse.
Beispiel in BASIC: `CALL &0000`

Adresse		Funktionsaufruf
0000h	-	Systemstart
0038 h	-	Interrupt-Register auslesen (ISR0 aufrufen)
0066 h	-	NMI aufrufen, erzeugt Neustart mit Kaltstart auf 0000h
0070 h	-	CPU-Temperatur auf COM1 senden
0074 h	-	Beep Ton ausgeben
0078h	-	Uhrzeit auf COM1 senden
007Bh	-	Datum auf COM1 senden
0080 h	-	Bildschirm löschen und Debugger-Menue starten
0084 h	-	IDE-Medium check Master auf IDE0
0088 h	-	BASIC Programm starten (wenn im SRAM vorhanden)
00FFh	-	CPU Halt auslösen >>> Alarm-Relais testen (rote LED auf CLK-Karte)

... weitere folgen später

Status von Konsole auf VGA-Monitor ausgeben

Um eine Statusmeldung der Konsolenverbindung ein-, oder auszublenden wurden zwei spezielle Direktbefehle in das Advanced-BASIC für die VGA-Konsole eingebunden:

- **XSTSON**

- **XSTSOFF**

Mit Eingabe des Befehls XSTSON wird die Meldung in einem reservierten Ausgabebereich des unteren Bildschirms erzeugt. Alle BASIC-Ein/Ausgaben können weiterhin gescollt werden, unabhängig der Statusmeldungen. Mit XSTSOFF wird die Meldung wieder gelöscht. Ein CLS- Befehl hat die gleiche Wirkung.

```
List

100 REM hohe MEM-Adresse anCALLen
110 CLS
120 :
130 XSEG(7)
140 XPOKE(&0010),201     :REM schreibe RET-Befehl nach 70010h
150 :
160 XSCALL(&0010),7      :REM springe Adresse 70010h an




Fertig

XSTSON↵
_
```

```
- SEPIA Terminal -
SEPIA VGA-Adapter 640 x 480 Pixel, 80 x 36 Character, 10 Color, VT-100 Emulation
BaudRate: 115.200 kBd  Data: 8 Bit  Stopp: 1 Bit  Parity: No  Handshake: RTS/CTS
```

Grafik-Befehle bei Verwendung der NET-Karte (mit VGA-Ausgang)

Auf den ersten Seiten wurden VGA-Grafik-Befehle nur kurz vorgestellt. Die Handhabung und Wirkungsweise dieser Advanced-BASIC-Befehle sollen nun näher erläutert werden. Da der SEPIA-Rechner in erster Linie für MSR- und Automatisierungszwecke dient, sind die Grafikfunktionen nur auf wenige Befehle beschränkt. Die Auflösung beträgt beispielsweise nur 640 x 240 Pixel mit 2 von 64 Farben. Auf Textausgaben und Kreisfunktionen wurde aus Platzgründen (im EEPROM) ebenfalls verzichtet. Die Kommunikation erfolgt hierzu auf COM1. Intern werden spezielle ESC-Sequenzen für Steuerkommandos verwendet, die der Grafik-Chip mit einem eigenen BIOS interpretiert. Somit wird kein SEPIA RAM-Speicher für den Bildaufbau verwendet und der Hauptprozessor vollkommen entlastet.

Beispiel: **XGRON**
Interne ESC-Sequenz: **ESC[z1**
Schaltet in den Grafik-Mode um. Der Bildschirm wird dabei gelöscht.

Beispiel: **XGROFF**
Interne ESC-Sequenz: **ESC[z2**
Schaltet in den Text-Mode um und inititalisiert den Bildschirm neu.

Beispiel: **XGCLS**
Interne ESC-Sequenz: **ESC[z0**
Löscht den internen Grafik-Bildspeicher (wie CLS).

Beispiel: **XGRTEST**
Interne ESC-Sequenz: **ESC[t@**
Erzeugt ein statisches Testbild als mehrstufige Farbtreppe auf dem Bildschirm.

Beispiel: **XGRESCOL**
Interne ESC-Sequenz: **ESC[z3**
Setzt die org. Farbpalette wieder zurück. (weiß auf blau)

Beispiel: **XGRSTCOL V,H**
Interne ESC-Sequenz: **ESC[<V><H>]**
Setzt Pixel-Farbe für Vordergrund und Hintergrund. (zwei-byte-kompliment)

Beispiel: **XGRPSET X,Y**
Interne ESC-Sequenz: **ESC[<XMSB><XLSB><Y>}**
Setzt einen Pixel an Koordinate X, Y (Xmax. = 640, Ymax. = 240 als zwei-word-kompliment)

Beispiel: **XGRLINE X1,Y1,X2,Y2**
Interne ESC-Sequenz: **ESC[<XMSB1><XLSB1><Y1><XMSB2><XLSB2><Y2>{**
Zeichnet eine Linie von Koordinate X1, Y1 zu X2, Y2. (vier-word-kompliment)

Grafik-Befehle bei Verwendung der NET-Karte (mit VGA-Ausgang) Teil2

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM Grafik-Test
110 XGRON                        : REM Grafik EIN
120 XGRTEST                      : REM Testbild
130 XDELAY(40000)                : REM Warten
140 XGRCLS                       : REM Grafik löschen
150 XGRSTCOL(255,2)              : REM Farbe setzen
160 XGRPSET(320,120)             : REM Pixel zeichnen
170 XGRLINE(10,100,630,100)      : REM Linie zeichnen
180 XGRLINE(20,200,630,110)      : REM Linie zeichnen
190 XDELAY(40000)                : REM Warten
200 XGROFF                       : REM Grafik AUS
210 STOP                         : REM Text-Mode
```

Hinweis:
Alle direkten Advanced-BASIC Ein/Ausgaben der Konsole erfolgen immer über COM1, welche durch DIP-Schalterstellungen von der CPU-Karte auf die NET-Karte (VGA-Ausgang und PS/2-Tastatur) umgelenkt werden können. Die Grafik erzeugt ein 8-Kern 32-Bit Parallax®-Prozessor mit 80 MHz interner Taktfrequenz. Die gesamte Datenübertragung erfolgt dazu über eine interne, serielle COM-Verbindung mit 115.200 kBaud als einfache Terminal-Emulation, wie zu einem abgesetzten LCD-Display. Dies ermöglicht ebenso den Einsatz von seriellen ANSI BBS Terminals oder Programmen mit VT-100 Emulation.

Diese Schaltungsanordnung hat somit mehrere Vorteile:

1) Der Hauptprozessor wird komplett mit der Erzeugung von Text und Grafik entlastet.

2) Die Datenübertragung kann auch zu anderen COM-Ports umgelenkt werden.
(bsp. per DFÜ MODEM oder Ethernet COM mit Windows ZOC-Programm)

3) Die Datenaufbereitung erfolgt in einem eigenen Prozess, unabhängig von CPU-Zeiten.
Damit sind die Ausgaben immer gleich schnell.

4) Die Auslagerung von Text- und Grafik-Routinen spart viel Platz im EPROM.

5) Die Weiterentwicklung von VGA Hard- & Software erfolgt unabhängig vom Hauptsystem.

6) Durch Nutzung gängiger Auflösungs-Modi sind nahezu alle VGA-Monitore verwendbar.
(Standardauflösung ist 640 x 480 Pixel, somit auch für 5“ LCD-Display geeignet)

EMail versenden mit XPort® (auf NET-Karte)

Damit SEPIA eine EMail versenden kann, wurde der Trigger-Eingang des XPort-Konverters mit dem CPLD über ein FlipFlop verknüpft. Mittels BASIC-Befehl wird darauf hin ein negativer Impuls an Trigger-Eingang (CP2) erzeugt. Werden die erforderlichen EMail-Parameter für den XPort über den HTML Browser programmiert (Grundkonfiguration), kann eine Nachricht (bsp. ALARM 123) über das Netzwerk abgesetzt werden. Zur Kontrolle wird ein „@“ Zeichen auf den jeweiligen COM-Port mit ausgegeben.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM EMail-Test
110 XSEMAIL 1                : REM sende auf COM1 = @
120 STOP
```

Wichtiger Hinweis zu XPort®
Damit eine EMail versendet werden kann, muss der Administrator neben der Server-IP Adresse auch den Domain-Namen nebst Unit-Namen fest eingeben. Der EMail-Empfänger ist variabel. Die Einstellungen sind dennoch nicht trivial. Zum bsp. müssen Einstellungen wie der Triggereingang „General Purpose I/O“ als Eingang mit Aktiv „Low“ parametriert - oder eine bestimmte Zeichenfolge für das Auslösen der EMail vorgenommen werden. Weitere Informationen finden Sie in der XPort-Beschreibung des Herstellers unter:
http://www.lantronix.com/support/downloads/?p=XPORT

Anmerkung:
Um erste TCP/IP Winsock Verbindungstests mit XPort durchzuführen, können Programme wie Hyperterminal oder Device-Terminal mit zugewiesener IP-Adresse verwendet werden. Seriell tunneln lassen sich die Daten jedoch nur mit einer Re-Direct-Software (bsp. CRP von Lantronix®), damit ein virtueller COM-Port die Pakete entsprechend umlenkt. Bevor jedoch irgend eine Verbindung zu stande kommt, müssen alle Grundparameter via HTTP-Browser voreingestellt werden. Die erste Konfiguration erfolgt somit i.d.R. über den Internet-Explorer. Als lokale „Webadresse“ nutzt man zunächst die Standard XPort IP: **http://192.168.1.11**

Für den EMail-Versand benötigt man zusätzlich einen Online-Zugang zu einem SMTP-Server mit automatischer Authentifizierung und fest einstellbarer IP-Adresse mit Port 25. Eine gute Lösung ist ein Modem-Router mit DSL-Zugang (bsp. Fritz Box) zu einem Provider, der SMTP ohne weitere Anmeldung erlaubt, welchen man quasi als Remote-IP Server nutzt.

Folgende Webseiten sind zu diesem Thema sehr hilfreich:
http://www.msxfaq.de/internet/inetanbindungsmtp.htm
http://www.albert-rommel.de/ppp-addrconf.htm
http://www.elektronik-kompendium.de/sites/net/0903081.htm
http://www.avm.de/de/Produkte/FRITZBox_an_jedem_Anschluss_Internet.html

XPULSE : Ein einfacher Befehl für Burst-/ Patterngenerator als Winkelgeberemulation

Mit diesem Befehl lassen sich abwechselnd verschiedene Logik-Pattern von 0...FFFF counts auf einer I/O-Adresse in Echtzeit erzeugen. Zur Ausgabe des Signals kann beispielsweise die LPT2 (auf &29h) oder AUX-Schnittstelle (auf &49h) mit TTL-Pegel verwendet werden. Die LPT2-Schnittstelle (mit 8255 Chip) wird bereits beim booten auf Ausgabe initialisiert, um beipielsweise einen zweiten Drucker anzusteuern. Somit lässt sich der Daten-Port (PB) auch für andere Zwecke verwenden, wenn auf diese I/O-Adresse direkt geschrieben wird.

Mit dem erzeugten Signal lassen sich beipielsweise programmierbare Bursts (bis zu 65535 Pulsen) mit versetzten Pegeln wie bei einem Drehgeber simulieren, oder einfach nur eine bestimmte Anzahl von Impulsen zu Prüfzwecken ausgeben. Wendet man den Befehl mehrmals hintereinander an, sind entsprechend höhere Counter-Bursts möglich. Während der XPULSE-Befehl ausgeführt wird, sollte die CPU keinen Interrupt verarbeiten.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

Beispiel Mode 0 (einfacher Impulsgenerator)

```
100 REM XPULSE-Test Port Toggle = 01010101 -> 10101010
110 XPULSE (&29),65535,0  : REM erzeugt jetzt 65535 Pulse
120 XDELAY (5000)         : REM kleine Pause
130 OUT (&29),0           : REM Port auf Null zurücksetzen
```

Beispiel Mode 1 (Drehgeber-Simulation B vor A Signal voreilend)

```
100 REM XPULSE-Test Bit0=A, Bit1=B, Bit2=/A, Bit3=/B
110 XPULSE (&29),1000,1   : REM erzeugt jetzt 1000 Pulse
120 XDELAY (5000)         : REM kleine Pause
130 OUT (&29),0           : REM Port auf Null zurücksetzen
```

Beispiel Mode 2 (Drehgeber-Simulation A vor B Signal nacheilend)

```
100 REM XPULSE-Test Bit0=A, Bit1=B, Bit2=/A, Bit3=/B
110 XPULSE (&29),2000,2   : REM erzeugt jetzt 2000 Pulse
120 XDELAY (5000)         : REM kleine Pause
130 OUT (&29),0           : REM Port auf Null zurücksetzen
```

Mode

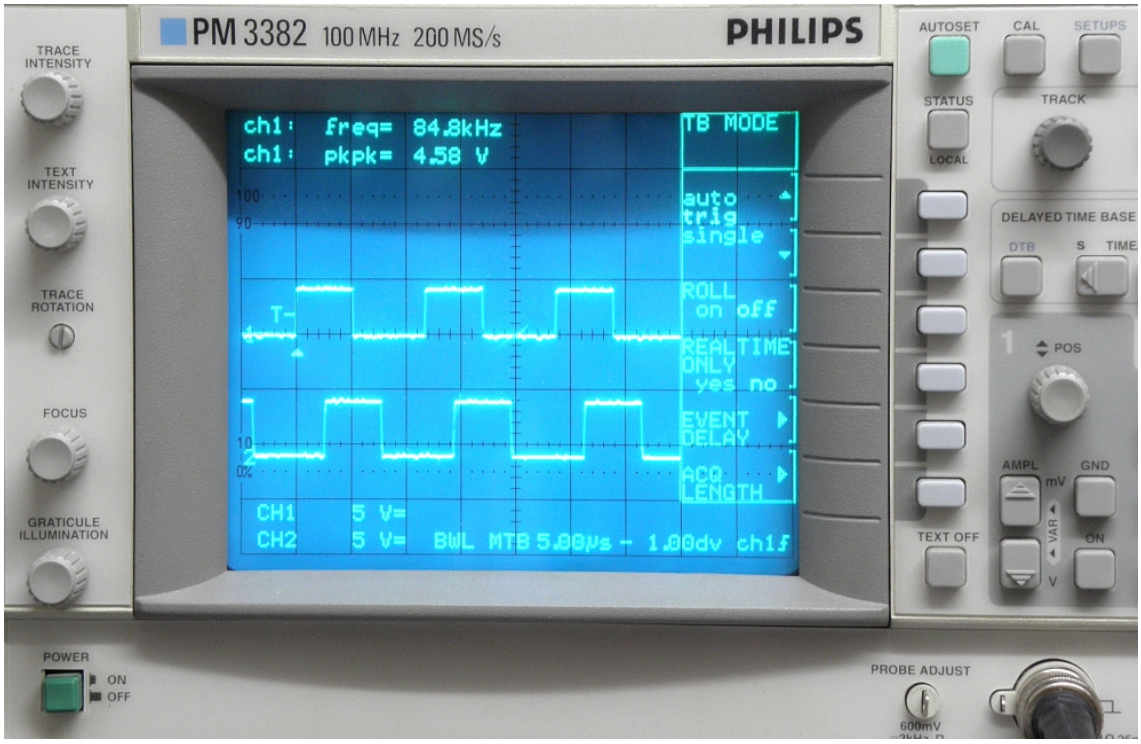
Hinweis zur Geschwindigkeit:
Signal-Taktfrequenz bei 20 MHz CPU mit Bus-Waitstate DIP-Schalter 1 = ON

Mode 0 = 150 kHz
Mode 1 = 85 kHz
Mode 2 = 85 kHz

Verwendet man Data Bit4 und Bit5 für Z und /Z, kann über den OUT-Befehl zusätzlich ein Reset für den empfangenden Quadraturempfänger programmiert werden.

XPULSE

Bild: Drehgeber-Oszillogramm Leitung Bit0 (A) und Bit1 (B) von LPT2 Datenport.



Befehl: XPULSE (&29),1000,2

XIPEAK : Einzel-Impuls mit variabler Dauer erzeugen

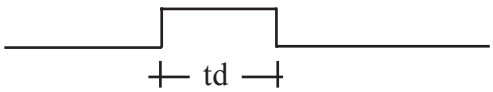
Dieser Befehl erzeugt auf einem TTL I/O-Port einen einzelnen Impuls mit variabler Dauer (Toggle-byte: 01010101 zu 10101010). Somit kann beispielsweise ein Puls-Störgenerator oder Trigger für Spannungsunterbrechungen (mit ext. Leistungsendstufe) realisiert werden. Die Dauer wird als 16-Bit Zahl übergeben, so dass die Impulszeit von ~5µs bis zu 85,2ms einstellbar ist. Mode wird zur Zeit mit 0 übergeben und dient für spätere Erweiterungen. Zur Ausgabe des Signals kann beispielsweise die LPT2 (auf &29h) oder AUX-Schnittstelle (auf &49h) mit TTL-Pegel verwendet werden. Die LPT2-Schnittstelle (mit 8255 Chip) wird bereits beim booten von SEPIA auf Ausgabe initialisiert, um beipielsweise einen zweiten Drucker anzusteuern. Somit lässt sich der Daten-Port auch für andere Zwecke verwenden, wenn auf diese Adresse direkt geschrieben wird. Während der XIPEAK-Befehl ausgeführt wird, sollte die CPU keinen Interrupt verarbeiten. Alle Impulszeiten ergeben sich bei 20 MHz CPU-Takt mit Waitstate DIP-Schalter 1 = ON (DIP-Switch auf Bus-Backplane).

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM XIPEAK-Test mit 1,3 ms Impuls
110 :
120 A = (&29)          : REM I/O-Adresse für TTL-Port
140 T = 1000           : REM 16Bit Variable für Impulszeit
160 M = 0              : REM Mode 0
180 :
200 XIPEAK (A),T,M     : REM Befehl ausführen
```

Die folgende Tabelle zeigt den Bezug bei Mode 0 zwischen Impulszeit und Einstelldauer:

Time (word)	~ Dauer	exakte Dauer (td)	
T = 1	5 µs	5,028 µs	(kleinste Verzögerung)
T = 2	6,5 µs	6,430 µs	
T = 5	10 µs	10,23 µs	
T = 10	17 µs	16,83 µs	
T = 20	30 µs	29,83 µs	
T = 50	69 µs	68,83 µs	
T = 100	133 µs	133,83 µs	
T = 767	1000 µs	1,000 ms	
T = 1000	**1,3 ms**	**1,3038 ms**	(wie im Beispiel angegeben)
T = 10000	13 ms	13,003 ms	
T = 16384	21 ms	21,202 ms	
T = 32768	42 ms	42,602 ms	
T = 65535	85 ms	85,199 ms	
T = 0	85 ms	85,200 ms	(längste Verzögerung)



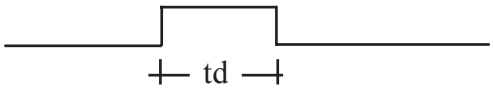
XIPEAK : Fortsetzung mit Mode = 1

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 REM XIPEAK-Test mit Mode = 1 für eine Sekunde Pulse
110 :
120 A = (&29)          : REM I/O-Adresse für TTL-Port
140 T = 312            : REM 16Bit Variable für Impulszeit
160 M = 1              : REM Mode 1
180 :
200 XIPEAK (A),T,M     : REM Befehl ausführen
```

Die folgende Tabelle zeigt den Bezug bei Mode 1 zwischen Impulszeit und Einstelldauer:

Time (word)	~ Dauer	exakte Dauer (td)	
T = 1	3,2 ms	3,205 ms	(kleinste Verzögerung)
T = 2	6,4 ms	6,406 ms	
T = 5	16 ms	16,010 ms	
T = 10	32 ms	32,017 ms	
T = 20	64 ms	64,032 ms	
T = 40	128 ms	128,060 ms	
T = 50	160 ms	160,074 ms	
T = 80	256 ms	256,117 ms	
T = 100	320 ms	320,145 ms	
T = 200	640 ms	640,286 ms	
T = 312	**1 Sekunde**	**998,843 ms**	(wie im Beispiel angegeben)
T = 625	2 Sek.	2000,99 ms	
T = 1000	3,2 Sek.	3201,41 ms	
T = 2000	6,4 Sek.	6402,82 ms	
T = 4000	12,8 Sek.	12805,64 ms	
T = 10000	32 Sek.	32014,09 ms	
T = 0	~ 210 Sek.		(längste Verzögerung)



Hinweis:
Dieser Befehl hat eine sehr hohe Priorität. Während der Ausführung zählt die CPU die programmierte Zeitschleife strikt herab. Somit können keine anderen Funktionen zeitgleich ausgeführt oder abgefragt werden (auch keine Tastatureingaben).

XIRQON und XIRQOFF - Interrupt

Diese Befehle veranlassen auf der **RTC-Karte** einen Interrupt-Generator mit 1 Sek. Takt-Impuls auszulösen, so dass auf der aktuellen Bus-Nummer ein Interrupt mit Vector 038h erzeugt wird, um beispielsweise eine spezielle ISR-Routine mit hoher Priorität direkt im EPROM oder SRAM anzusteuern.

Die Schaltung (ATMEL-Prozessor) erzeugt nach Freigabe einen Puls mit 1000 ms Dauer, bestehend aus 1 µs neg. Puls + 999,999 ms pos. Puls-Länge, so dass im Endeffekt eine Periodizität von einer Sekunde erreicht wird.

Geben Sie das nachfolgende BASIC-Programm ein und starten Sie mit **RUN**

```
100 XIRQON                : REM Generator einschalten
110 XDELAY (50000)        : REM Zeitschleife
120 XIRQOFF               : REM Generator ausschalten
130 STOP                  : REM Programmende
```

Hinweis:

Nach Zeile 100 erfolgt nun eine sekündliche Konsolenausgabe mit: **IRQ: xx**
(xx Steht für Bus-Steckplatz 1...14), bis die Delay-Schleife in Zeile 110 abgearbeitet ist.

Mit XIRQOFF wird in Zeile 120 der Interrupt-Generator wieder abgeschaltet (disable) und anschließend mit Zeile 130 das Programm ganz normal beendet.

Der Anwender hat nun die Möglichkeit, über eine Assembler-Programmierung mittels neuer Sprungadresse (an 038h im EPROM) eine eigene ISR-Routine im oberen SRAM-Bereich oder im oberen EPROM-Bereich (7FF0-7FFEh) zu hinterlegen. (Programmer erforderlich)

Durch den erzeugten Interrupt vergrößert sich das Zeitverhalten der hier verwendeten Delay-Schleife nur marginal um die Ausgabedauer der erzeugten Meldung auf dem Bildschirm. Diesen Umstand muss man bei der Programmierung stets mit berücksichtigen, wenn ereignis- und/oder zeitkritische Unterprogramme über Interruptanforderungen abgearbeitet werden sollen.

Hardwaretechnisch befindet sich das Enable-Bit auf der I/O-Adresse 02Eh (Data 0).
Mit XIRQON wird gleichzeitig ein EI Befehl zur CPU abgesetzt. Es besteht zusätzlich die Möglichkeit noch weitere Interruptanforderungen abzuarbeiten, wenn das IRQ-Register in die Programmierung einbezogen wird. >> IRQ-Register siehe MAPALL.ASM

Weitere BASIC-Beispiele

```
100 REM Kreisumfang berechnen
110 INPUT „Radius“;R
120 U=2*PI*R
130 F=PI*R*R
140 PRINT „RADIUS=“;R
150 PRINT „UMFANG=“;U
160 PRINT „FLÄCHE=“;F

100 REM Cosinuns
110 PRINT COS(PI)
130 D=180
140 R=D*PI/180
150 PRINT COS(R)

100 REM String Zeichenkette bearbeiten
110 A$="Test123"
120 B$=LEFT$(A$,3)
130 PRINT B$

100 INPUT"Wort,Länge";A$,L
110 IF A$="E"THEN END
120 A$=FIX$(A$,L)
130 PRINT A$;"Länge";LEN(A$)

100 REM IDE-Medien scannen mit "DRIVE S" Befehl
110 CLS
120 LOCATE 2,2
130 DRIVE S

100 REM auf Taste warten
110 XWAITKEY
120 PRINT "o.k."

100 REM ASCII-Zeichencode zeigen
110 CLS
120 LOCATE 2,2
130 B = ASC(INKEY$)
140 PRINT "ZEICHEN "; CHR$(B); "  DEZ.:";B
150 IF B = 5 THEN STOP ELSE GOTO 130

100 INPUT "GEBEN SIE EINE ZAHL EIN", A
110 LET B=A*A
120 PRINT "DAS QUADRAT VON "; A; " IST "; B

100 INPUT "GEBEN SIE EINE POSITIVE ZAHL EIN", A
110 IF A>=0 THEN 140
120 PRINT "DIE ZAHL "; A; "IST NEGATIV!"
130 GOTO 100
140 PRINT "DIE WURZEL VON "; A; " IST "; SQR( A )

100 INPUT "LAENGE DER 1. KATHETE:", A
110 INPUT "LAENGE DER 2. KATHETE:", B
120 C = SQR( A*A + B*B )
130 PRINT "LAENGE DER HYPOTENUSE: "; C
```

Steuerzeichen lt. ASCII Tabelle

```
00 - NUL   Null character (Nullzeichen)
01 - SOH   Start of heading (Anfang Dokumentkopf)
02 - STX   Start of text (Textanfang)
03 - ETX   End of text (Textende)
04 - EOT   End of transmission (Ende der Übertragung)
05 - ENQ   Enquiry (Anfrage)
06 - ACK   Acknowledgment (Bestätigung)
07 - BEL   Audible bell (Tonsignal)
08 - BS    Backspace (Rückschritt)
09 - HT    Horizontal tab (Horizonaltabulator)
10 - LF    Line feed (Zeilenvorschub)
11 - VT    Vertical tab (Vertikaletabulator)
12 - FF    Form feed (Seitenvorschub)
13 - CR    Carriage return (Wagenrücklauf)
14 - SO    Shift out (Dauerumschaltung)
15 - SI    Shift in (Ende Dauerumschaltung)
16 - DLE   Data link escape (Datenverbindungs-Fluchtsymbol)
17 - DC1   Device control 1 (Gerätekontrollcode 1)
18 - DC2   Device control 2 (Gerätekontrollcode 2)
19 - DC3   Device control 3 (Gerätekontrollcode 3)
20 - DC4   Device control 4 (Gerätekontrollcode 4)
21 - NAK   Negative acknowledgment (Negative Bestätigung)
22 - SYN   Synchronous idle (Synchronisationssignal)
23 - ETB   End of transmission block (Ende des Übertragungsblockes)
24 - CAN   Cancel (Abbruch)
25 - EM    End of medium (Ende des Mediums)
26 - SUB   Substitute character (Ersetzen)
27 - ESC   Escape (Fluchtzeichen)
28 - FS    File separator (Dateitrenner)
29 - GS    Group separator (Gruppentrenner)
30 - RS    Record separator (Datensatztrenner)
31 - US    Unit separator (Einheitentrenner)
127 - DEL  Delete (Entfernen/Löschen)
```

ASCII-Code-Tabelle der ISO-8859-Familie (Byte-Werte 00 bis 127)

00	NUL	**01**	SOH	**02**	STX	**03**	ETX	**04**	EOT	**05**	ENQ	**06**	ACK	**07**	BEL
08	BS	**09**	HT	**10**	LF	**11**	VT	**12**	FF	**13**	CR	**14**	SO	**15**	SI
16	DLE	**17**	DCI	**18**	DC2	**19**	DC3	**20**	DC4	**21**	NAK	**22**	SYN	**23**	ETB
24	CAN	**25**	EM	**26**	SUB	**27**	ESC	**28**	FS	**29**	GS	**30**	RS	**31**	US
32	SP	**33**	!	**34**	„	**35**	#	**36**	$	**37**	%	**38**	&	**39**	‘
40	(	**41**	)	**42**	*	**43**	+	**44**	,	**45**	-	**46**	.	**47**	/
48	0	**49**	1	**50**	2	**51**	3	**52**	4	**53**	5	**54**	6	**55**	7
56	8	**57**	9	**58**	:	**59**	;	**60**	<	**61**	=	**62**	>	**63**	?
64	@	**65**	A	**66**	B	**67**	C	**68**	D	**69**	E	**70**	F	**71**	G
72	H	**73**	I	**74**	J	**75**	K	**76**	L	**77**	M	**78**	N	**79**	O
80	P	**81**	Q	**82**	R	**83**	S	**84**	T	**85**	U	**86**	V	**87**	W
88	X	**89**	Y	**90**	Z	**91**	[	**92**	\	**93**	]	**94**	^	**95**	_
96	‘	**97**	a	**98**	b	**99**	c	**100**	d	**101**	e	**102**	f	**103**	g
104	h	**105**	i	**106**	j	**107**	k	**108**	l	**109**	m	**110**	n	**111**	o
112	p	**113**	q	**114**	r	**115**	s	**116**	t	**117**	u	**118**	v	**119**	w
120	x	**121**	y	**122**	z	**123**	{	**124**	|	**125**	}	**126**	~	**127**	DEL

CTRL - ASCII character table

Dec	Hex	ASCII	Function	Key	SEPIA-BASIC	CP/M
0	00	NUL	(null)	CTRL @		
1	01	SOH	(start of heading)	CTRL A	x	x
2	02	STX	(start of text)	CTRL B		x
3	03	ETX	(end of text)	CTRL C	x	x
4	04	EOT	(end of transmission)	CTRL D		x
5	05	ENQ	(enquiry)	CTRL E	x	x
6	06	ACK	(acknowledge)	CTRL F		x
7	07	BEL	(bell)	CTRL G	x	x
8	08	BS	(backspace)	CTRL H		x
9	09	HT	(horizontal tab)	CTRL I		x
10	0A	LF	(line feed)	CTRL J		x
11	0B	VT	(vertical tab)	CTRL K		x
12	0C	FF	(form feed)	CTRL L		x
13	0D	CR	(carriage return)	CTRL M		x
14	0E	SO	(shift out)	CTRL N		x
15	0F	SI	(shift in)	CTRL O		x
16	10	DLE	(data link escape)	CTRL P	x	x
17	11	DC1	(device control 1)	CTRL Q		x
18	12	DC2	(device control 2)	CTRL R		x
19	13	DC3	(device control 3)	CTRL S		x
20	14	DC4	(device control 4)	CTRL T		x
21	15	NAK	(negative acknowledge)	CTRL U		x
22	16	SYN	(synchronous idle)	CTRL V		x
23	17	ETB	(end of transmission blk)	CTRL W		x
24	18	CAN	(cancel)	CTRL X		x
25	19	EM	(end of medium)	CTRL Y		x
26	1A	SUB	(substitute)	CTRL Z		x
27	1B	ESC	(escape)	CTRL [	x	x
28	1C	FS	(file separator)	CTRL \		
29	1D	GS	(group separator)	CTRL [		
30	1E	RS	(record separator)	CTRL ^		
31	1F	US	(unit separator)	CTRL _		

* ESC gefolgt von weiteren Steuerzeichen dient für formatierte Print-, Cursor- u.a. Bildschirmausgaben.

Die ersten 32 Steuerzeichen sind „nicht-druckend“, d.h. sie dienen bei der Ausgabe nur zu Steuerzwecken, beispielsweise zur Übertragung von X,Y-Koodrinaten bei der Grafikerzeugung, oder bei der Cursorsteuerung, um Zeichen an einer bestimmten Stelle auf dem Display zu positionieren bzw. darzustellen.

Die Kodierung von ASCII-Zeichen erfolgt bei SPEIA mit 7 bit. Das 8te bit wird intern für zusätzliche Aufgaben und Steuersequenzen verwendet. Die Zuordnung der ersten (0...127) Zeichen bzw. 128 Bitkombinationen ist gem. ASCII (American Standard Code for Information Interchange) entsprechend der **ASCII-Code-Tabelle** festgelegt.

Der Tastatur-Prozessor

Übersicht der vorprogrammierten F-Tastenbelegung

Kürzel	Befehl	
F1	LIST	Die beiden F-Tasten F1 und F2 werden mit CR & LF
F2	RUN	abgeschlossen, so dass bei betätigen der Taste der
F3	LOAD	Befehl (ohne RETURN) unmittelbar ausgeführt wird.
F4	SAVE	
F5	Renumber 100,20	
F6	PRINT FRE(X)	
F7	LOADGO	
F8	DIR	(für CP/M reserviert) - erzeugt Direktory
F9	USER 0	(für CP/M reserviert) - schaltet auf Haupt-User 0 um
F10	STAT DSK:	(für CP/M reserviert) - Disk Statistik ausgeben
F11	STAT VAL:	(für CP/M reserviert) - Disk Statistik ausgeben
F12	RESET	(für CP/M reserviert) - ruft Diensprogramm RESET.COM auf
Shift F1	XSTSON	schaltet Terminal-Status-Parameter in Zeile 37 - 40 EIN
Shift F2	XSTSOFF	schaltet Terminal-Status-Parameter in Zeile 37 - 40 AUS (bei CP/M über Dienstprogramme)

Falls eine PS/2-Tastatur an der NET-Karte betrieben wird, welche eine „Windows-Menue“ Taste besitzt, kann hiermit das Advanced-BASIC unmittelbar mir „Bye“ verlassen werden. Anschließend wird das Debugger-Menue für weitere Eingaben aufgerufen. Die „Druck“-Taste leitet ein komplettes BASIC-Listing (falls im Speicher vorhanden) auf einen angeschlossenen Drucker an LPT1 um.

Der manuelle Befehl lautet dazu: LIST#2 Die „Rollen“-Taste führt ein CTRL-R aus. Hiermit werden bisherige Eingaben aus dem Zeilen-Buffer kopiert und ein weiteres mal zur Ausgabe wiederholt.

Die „Pause/Unterbr.“-Taste kann während eines Listings die Ausgabe auf dem Monitor an der aktuellen Zeilennummer „einfrieren“ und bei wiederholter Betätigung das Listing fortführen. Sie hat die gleiche Funktion wie die Space-Taste, welche auch zur Pause bzw. als Unterbrechung für laufende Programmanwendungen genutzt werden kann. CTRL-E unterbricht ein laufendes Programm an der aktuellen BASIC-Zeilenummer welche gerade vom Interpreter im Speicher bearbeitet wird. Mit „CONT“ kann das Programm mit der darauf folgenden Zeilennummer wieder fortgeführt werden, falls keine Änderungen am Programmtext vorgenommen wurden. Alle Zwischenwerte (Variablen) bleiben erhalten. Die „TAB“-Taste führt einen Cursor-Sprung (Tabulator) um 8 Zeichen nach rechts aus. Die „Shift-TAB“-Tastenkombination macht die letzte TAB-Eingabe wieder rückgängig.
Über die „AltGR“-Taste sind viele Sonderzeichen der dritten Tastenebene erreichbar.

Die „Caps-Lock“-Taste (Pfeil nach unten) ist bei SEPIA als Umschalter definiert. Sie schaltet bei Betätigung jeweils von Klein-nach-Großbuchstaben, oder Groß-nach-Kleinbuchstaben um. Wird die „Shift“-Taste (Pfeil nach oben) vor einem zweiten Tastendruck mitgewählt, kann vorübergehend von Klein-nach-Großbuchstaben manuell umgeschaltet werden, so lange die Taste gehalten wird.

Hinweise zum Scan-Code der Tastatur:
- Die NUM-Lock Taste wird beim Booten nicht aktiviert. Damit sind AT-Pfeiltasten sofort nutzbar.
- CTRL-Zeichen erfolgen unter CP/M (optional) mit ^ vor dem empfangenen Buchstaben.
- Die Zeichen **§** und **°** (deutsches Layout) werden gem. ASCII-Tabelle nicht unterstützt.
- Die deutschen Umlaute **ä, ö, ü** werden direkt als Doppelzeichen in **ae, oe** und **ue** übersetzt.
- Der Buchstabe **ß** wird in **ss** übersetzt.

Web - Links zu technischen Dokumentationen im PDF-Format:

Info-Flyer	http://www.kolter.de/SEPIA_Flyer01.pdf
SEPIA Datenblatt	http://www.kolter.de/SEPIA_dblatt01.pdf
Übersicht Komponenten	http://www.kolter.de/SEPIA-Konfiguration.pdf
Stromverbrauch	http://www.kolter.de/SEPIA-Stromverbrauch.pdf
CE-Erklärung	http://www.kolter.de/ce-sepia.pdf
SEPIA-Bus Platine	http://www.kolter.de/SEPIA-BUS.pdf
CPU	http://www.kolter.de/SEPIA-CPU.pdf
IDE	http://www.kolter.de/SEPIA-IDE.pdf
NET (inkl. VGA)	http://www.kolter.de/SEPIA-NET.pdf
MEM	http://www.kolter.de/SEPIA-MEM.pdf
RTC (inkl. WatchDog)	http://www.kolter.de/SEPIA-RTC.pdf
CLK	http://www.kolter.de/SEPIA-CLK.pdf
RL8 - Relais	http://www.kolter.de/SEPIA-RL8.pdf
OI8 - Opto Input	http://www.kolter.de/SEPIA-OI8.pdf
AD1 - Analog Input	http://www.kolter.de/SEPIA-AD1.pdf
CNT - Counter	http://www.kolter.de/SEPIA-CNT.pdf
E32 - Opto Input	http://www.kolter.de/SEPIA-E32.pdf
A32 - Opto Output	http://www.kolter.de/SEPIA-A32.pdf
DA1 - DAC Ausgang	http://www.kolter.de/SEPIA-DA1.pdf
PW1...7 DC-Wandler	http://www.kolter.de/SEPIA-PWx.pdf
Netzteil - Datenblatt	http://www.kolter.de/GS120-spec.pdf
- - -	
Foto, 42 TE Gehäuse	http://www.kolter.de/TG42-seitlich-002a.jpg
Foto, 63 TE BGT	http://www.kolter.de/SEPIA_Bild7.jpg
Foto, Frontplatte CLK	http://www.kolter.de/ZMCBTL-001.jpg

Diese Seite bleibt frei !

Kapitel 2

SEPIA
Standard-BASIC Befehle

Vorwort

Neben den Advanced-Befehlen, welche spezielle SEPIA-Funktionen und dessen Hardware unterstützen, sind auch „normale“ BASIC-Befehle zur Programmierung anwendbar.

BASIC (**B**eginners **A**ll-purpose **S**mbolic **I**nstruction **C**ode) wurde am Dartmouth College von John Kemeny und Thomas Kurtz als Lernmittel grundlegender Programmierkonzepte entwickelt. Seitdem hat sie als zeitsparende und universelle Programmiersprache weltweit Anerkennung erlangt. Sie wird auch heute noch als eine der am leichtesten Hochsprachen betrachtet. Seit 1964 gibt es bis Dato über hundert verschiedene „Ableger“. Diese sind in ihrer bestechend einfachen Syntax sehr schnell erlernt und leicht auf jeden Rechner anwendbar.

Dieses Kapitel enthällt nun eine alphabetische Auflistung der Standard BASIC-Instruktionen, welche mit SEPIA genutzt werden können. Dazu werden die Befehle bzw. deren Funktion in Syntax, Wirkungsweise und Beispiele untergliedert.

Die Instruktionen werden wie folgt beschrieben:

Verwendungszweck	=	Funktion, zweck einer Anweisung
Syntax	=	Vollständige Notation der Instruktion
Wirkungsweise	=	Nützliche Information über die Instruktion. Beschreibung, welche Auswirkungen dieser Befehl während seiner Ausführung bewirkt
Beispiele	=	Programmbeispiel, wie der Befehl in einem Programm angewendet wird
Hinweise	=	Spezielle Hinweise zum Befehl

Befehlsübersicht: SEPIA Standard-BASIC

A:
ABS
AND
ASC
AUTO
ATN

B:
BYE

C:
CHR
CLEAR
COS
CALL
CONT

D:
DATA
DIM

E:
EE
EDIT
ELSE
END
ERL
ERR
ERR1
ERROR
EQV
EXP
EXIT

F:
FOR
FRE
FIX
FIX$
FIND

G:
GOTO
GOSUB

H:
HEX

I:
INP
INPUT
IF
INKEY$
INSTR
INT
IMP

K:
KILL

L:
LET
LVAR
LIST
LOG
LOG10
LEN
LEFT$
LOCATE

M:
MID$
MOD

N:
NEW
NEXT
NOT

O:
OUT
OUTBYTE
ON
OR

P:
PI
POKE
POS
PEEK
PRINT
PRECISION
PRIVACY

R:
READ
RESTORE
RETURN
RANDOMIZE
REM
RESUME
RENUMBER
RUN
RND
RND1
RIGHT$
REPLACE

S:
STOP
SIN
SGN
SQR
STR$
STRING$
STEP
SPC

T:
TAN
TRACE
TO
THEN
TAB

U:
USING

V:
VAL
VARPTR

W:
WAIT

X:
XOR

BASIC Fehlermeldungen:

(nach Fehlernummer sortiert)

1 NEXT ohne FOR
2 Syntaxfehler
3 RETURN ohne GOSUB
4 keine Daten
5 nicht erlaubt
6 Ergebnis zu gross
7 kein Speicherplatz
8 Sprungziel fehlt
9 DIM-Befehl falsch oder fehlt
10 DIM-Befehl doppelt
11 durch 0 teilen ?
12 kein Direktbefehl
13 Datentypen (Zahl/String) vermischt
14 kein Stringplatz
15 String zu lang
16 Ausdruck zu schwierig
17 CONT geht nicht
18 USER FN CALL fehlt
19 Dateiende falsch
20 Zeilen-Nr. falsch
21 BASIC verfügbar
22 FNRETURN ohne FN CALL
23 Zeilen-Nr. fehlt
24 RECORD zu lang
25 MATRIX nicht definiert
26 Kanal falsch
27 RESUME ohne ERROR
28 Inhaltsverzeichnis voll
29 Extension falsch
30 DISK voll
31 Datei nicht gefunden
32 Datei existiert
33 Datei geschlossen
34 OPEN-Modus falsch
35 Dateiname falsch
36 OPTION falsch
37 INPUT nicht offen
38 OUTPUT nicht offen
39 Wahlzugriff geht nicht
40 Datei nicht offen
41 keine Binäreinheit
42 kein Dateiplatz
43 RECORD-Nr. falsch
44 DISK geschützt
45 EXIT ohne FOR
46 nicht verfügbar
47 nicht vorhanden
48 kein internes Format
49 Fehler unbekannt
50 Fehler unbekannt

Befehlsform: ERROR (Fehlernummer)
Wie bei den Direktbefehlen, kann mit dem Befehl ERROR (0-50) eine Fehlermeldung direkt ausgegeben werden. Fehlermeldungen sind aber auch beliebig definierbar, wobei die Texte intern den Nummern 0...255 zugeordnet sind. Entsprechend der gewünschten Abbruchbedingung des Programmes wird die Fehlernummer dann bei Erreichen dieser Bedingung ausgegeben.

```
10 LET A = 6
20 IF A >5 THEN ERROR 50
RUN
Ausgabe: Fehler unbekannt Zeile 20
```

Während das Programm auf die Zeile 20 trifft und der Wert der Variablen A größer 5 ist, wird die Zahl 50 als Fehlernummer gespeichert und zusammen mit der Abbruchzeile nach RUN ausgegeben.

Systemfehlermeldungen:

* Falsche Eingabe*
* Komma falsch *
** HALT **
** HALT ** Zeile ZE nicht gefunden * Sprungziel fehlt *

Operatoren

=	Zuweisung bzw. ist gleich	
<	kleiner als	
>	größer als	
<=	kleiner oder gleich	bzw. =< gleich oder kleiner
>=	größer oder gleich	bzw. => gleich oder größer
<>	ungleich	
()	Ausdruck in Klammern	
^	Exponentialoperator (Hochzahl, oder auch Potenzierung)	
-	Negation (Zahl oder Ausdruck)	
*	Multiplikation	
/	Division	
-	Subtraktion	
\	Ganzzahl-Division („mit Rest“)	
MOD	modulo (Der „Rest“ bei der Division zweier Variable wird als Ergebnis zurückgegeben)	
NOT	binäre Negation	
AND	logische UND-Verknüpfung	
OR	logische ODER-Verknüpfung	
XOR	logische Exklusiv-Oder-Verknüpfung	
EQV	logische Gleichwertigkeit	
IMP	logische Ungleichheit (Implizierung)	

Bei Operatoren in einer Zeile nimmt die Priorität von links nach rechts ab. Alle logische Operationen setzen ihre Operatoren in 16-Bit-Ganzzahlwerte um. Dabei ist der Bereich von 0 - 65535 (FFFF) oder von -32768 bis 32767 erlaubt (sonst Fehlermeldung: nicht erlaubt).

Mathematische Standard-Funktionen

SIN(X)	Sinus (Argument im Bogenmaß)
COS(X)	Cosinus (Argument im Bogenmaß)
TAN(X)	Tangens (Argument im Bogenmaß)
ATN(X)	Arcus Tangens (im Bogenmaß)
SQR(X)	Quadratwurzel
EXP(X)	Exponentialfunktion
LOG(X)	natürlicher Logarithmus
ABS(X)	Absolutbetrag
INT(X)	Ganzzahl <= x
SGN(X)	Vorzeichen (-1, 0 oder 1)
RND(0)	Pseudo-Zufallszahl (0 <= y < 1)

Die Klammer nach dem BASIC-Befehl dient als Hinweis, daß immer ein Ausdruck in Klammern folgen muß, der auch durch eine numerische Variable oder durch einen arithmetischen Ausdruck ersetzt sein kann. Der berechnete Wert kann einer nummerischen Variablen zugewiesen werden. Bei Winkelfunktionen muß der Winkel in Bogenmaß (rad) angegeben werden. Die Umrechnung erfolgt durch Division des Winkelwertes (Grad = Altgrad) mit 180 und Multiplikation mit der Zahl Pi (1 Grad entspricht 0,017453.. rad). Die Variablen EE und PI enthalten fest gespeichert die Zahlenwerte für die Eulersche Zahl e und die Kreiszahl Pi.

Wird eine Division mit diesem BASIC-Befehl durchgeführt, so entfällt die Verwendung des INT(-Befehls), wenn ein ganzzahliges Ergebnis benötigt wird. Beispiel: PRINT 5 DIV 3 <CR> Anzeige: 1

Der Befehl **RANDOMIZE** ergibt einen zufälligen Startpunkt der Pseudo-Zufallszahlenfolge aus dem R-Register.

RND: Wird als Argument der Wert Null angegeben, wird erstmalig eine beliebige Zahl angezeigt und dann wiederholt. Beispiel: PRINT RND(0) <CR> Anzeige z.B.: .5019531548

Werte größer Null entsprechen dem RND-Befehl ohne Argument. PRINT RND(1) ergibt die selben Zahlen wie PRINT RND. Negative Argumente bestimmen den Startpunkt der Zufallszahlenfolge.

Variablen

Variablen sind in der Mathematik weit verbreitet und dienen meißt als symbolische Platzhalter numerischer Inhalte. Das Standard-BASIC verwendet hier nur globale Variablen. Diese werden in der Regel nicht deklariert, sondern sind direkt bei Verwendung (durch Überladen) verfügbar. Der Befehl „LET“ weist einen Wert einer Variablen regulär als Ausdruck zu. Ein Ausdruck kann eine Konstante, eine andere Variable, oder ein zuvor berechnetes Zwischenergebnis sein.

Ein einfaches BASIC-Beispiel:

```
10  LET A = 1.05
20  LET B = 3.30
30  C = A + B
50  PRINT C
Run
Ausgabe: 4.35
```

Vorsicht ist jedoch bei Nutzung von bereits vergebenen Schlüsselwörtern wie BASIC-Befehlen, oder Festwerten wie beispielsweise „Pi“ geboten. EE ohne Argument enthält die Eulersche Zahl e, die Basis der natürlichen Logarithmen. Beispiel: PRINT EE <CR> Anzeige: 2.7182818285

Alle Variablen bleiben nach Beendigung, oder Unterbrechung, des Programms verfügbar. Dadurch gestaltet sich eine Fehlersuche beim Debuggen der Variablen-Inhalte recht einfach. Bei Änderung des Quelltextes, oder neuen Programmzeilen, werden Variablen-Inhalte jedoch wieder zerstört. Mit dem BASIC-Befehl „CLEAR“ kann man ebenfalls sämtliche Variablen löschen. Die Inhalte werden mit Null überschrieben.

Ein einfaches BASIC-Beispiel:

```
10  A = 123
20  B = 456
30  CLEAR
40  C = 789
50  PRINT A,B,C
Run
Ausgabe: 0 0 789
```

Variablentypen

Das Standard-BASIC kennt in seiner Grundform nur zwei Variablentypen. Eine normale Variable ist numerisch, in den meisten Fällen ein Byte, Integer-Wert, oder eine Fliesskommazahl. Der andere Typ ist eine Textvariable. Diese wird im BASIC-Quelltext durch ein angehängtes String-Zeichen ($) kenntlich gemacht. Der hinterlegte Dateninhalt einer String-Variable enthält somit ein Character-Zeichen und ist daher byte-orientiert strukturiert. (dezimal 0...255)

Ein einfaches BASIC-Beispiel:

```
10  A$ = „Hallo! “
20  B$ = „ Heute“
30  C$ = „ ist einer schoener“
40  D$ = „ Tag.“
50  PRINT A$;B$;C$;D$
Run
Ausgabe: Hallo! Heute ist ein schoener Tag.
```

ABS

Verwendungszweck:
Liefert den absoluten Wert des Ausdrucks n. Damit wird der Absolutwert, also der Wert ohne Vorzeichen, einer Zahl ermittelt. Der Absolutwert einer negativen Zahl ist eine Zahl multipliziert mit -1. ABS kann sowohl auf Zahlen als auch auf komplexe numerische Ausdrücke angewendet werden.

Syntax:
ABS(n)

Wirkungsweise:
n muss ein numerischer Ausdruck sein

Beispiel:
```
10 PRINT ABS(-1)
20 IF ABS(N)=N THEN PRINT „POSITIVE ZAHL ODER NULL“
```

AND

Verwendungszweck:
Mathematische Operation einer logischen UND-Verknüpfung um Bit-Folgen zu testen.
AND wird bespielsweise zum Maskieren von numerischen Daten verwendet.

Syntax:
(n1) AND (n2)

Wirkungsweise:
n muss ein numerischer Ausdruck sein

Beispiel:
```
10 X =  63 AND 16
20 PRINT X

RUN
```

Hier wird X auf den Wert 16 gesetzt. Da 63 = binär 00111111 ergibt und 16 = binär 10000 ist, ergibt 63 AND 16 binär = 010000, und somit wieder 16.

ASC

Verwendungszweck:
Erzeugt einen numerischen Wert, der den ASCII Code für das erste Zeichen einer String$-Zeichenfolge enthält.

Syntax:
ASC(X$)

Wirkungsweise:
Wenn das Ergebnis X$ Null ist, erfolgt eine Fehlermeldung.
Die ASCII-Zahlen 0...9 bewirken nacheinander die Ausgabe 48....57.
Weitere Werte-Zuordnungen können der ASCII-Tabelle entnommen werden.

Beispiel:
```
10 X$ = „1Hallo“
20 PRINT ASC(X$)

RUN
```

Nach Starten der Programmzeile 10 wird der Wert 49 (für 1)
als Print-Ergebnis auf den Bildschirm ausgegeben.

ATN

Verwendungszweck:
Das Ergebnis liefert den Arcustangens von X, wenn X im Bogenmaß angegeben wurde.

Syntax:
ASC(X)

Wirkungsweise:
Der berechnete Wert liegt im Bereich von -π/2 ...bis.... π/2.
Der Ausdruck X kann ein beliebiger, numerischer Wert sein. Die Umrechnung mit ATN erfolgt mit einfacher Genauigkeit, max. jedoch mit 12 Stellen.

Beispiel:
```
10 INPUT X
20 PRINT ATN(X)
```

RUN

Nach Starten der Programmzeile 10 und Input-Eingabe von „3“ wird der Arcustangens von X berechnet. Das Ergebnis liefert den Wert mit 1.249046 zurück.

Hinweis:
Der Wert als Variable „Pi“ ist bereits als Festwert im ROM zugewiesen. Pi kann somit nicht anders belegt oder überschrieben werden.

AUTO

Verwendungszweck:
Dieser Befehl gibt automatisch Zeichennummern für Programmzeilen aus. Erste Zeichennummer und Differenz zwischen den Zeichennummern können zusätzlich angegeben werden.

Syntax:
AUTO [Zeilennummer][,[Schrittweite]]
AUTO .[,[Schrittweite]]

AUTO <CR>
Es wird mit Zeichennummer 10 begonnen. Nach jedem nächsten <CR> wird zur Zeichennummer die Zahl 10 addiert und diese ausgegeben.

AUTO 100 <CR>
Es wird mit Zeichennummer 100 begonnen und nach jedem <CR>10 dazu addiert.

AUTO 10,100 <CR>
Es wird mit Zeichennummer 10 begonnen, aber nach jedem <CR> wird die Zahl 100 dazu addiert. Wird die Zeichennummer 65535 erreicht, so erfolgt die Fehlermeldung „Zeichen-Nr.falsch“.

AUTO 10+,100 <CR>
Es wird der Zeichenabstand zum Anfangswert addiert und mit der Summe begonnen. Existiert schon eine durch AUTO erhaltene Zeichennummer, wird diese mit einem vorangesteltem + Zeichen ausgegeben. <CR> läßt die dazugehörige Programmzeile unverändert. Sie kann durch eine neue überschrieben werden.

<CTRL-E>
beendet die automatische Zeichennummerierung. BASIC kehrt wieder zur ursprünglichen Befehlsebene zurück.

Wird AUTO nachher mit AUTO. (Punkt eingeben) wieder begonnen, so werden die Zeichennummern beginnend mit der letzten Nummer weitergeführt, wobei die Differenz wieder angegeben werden kann, z.B.: AUTO .,20 (bsp. für Differenz = 20).

Beispiel:
AUTO 100,50
Generiert die Zeilennummer 100, 150, 200 usw.

AUTO
Generiert die Zeilennummer 10, 20, 30, 40 usw.

BEEP

Verwendungszweck:
Erzeugen eines kurzen Quittungs-Ton auf der CPU-Karte.

Syntax:
BEEP <CR>

Wirkungsweise:
Bei „BEEP“ erzeugt eine kleine Unterroutine auf dem 8255-Port ein Rechteck-Signal.

Beispiel:
10 REM Test
20 BEEP

RUN

Bei Erreichen der Zeile 20 wird ein kurzer Audio-Ton erzeugt. Die Ausgabefrequenz und Dauer ist abhängig von CPU-Takt und eingestellten Waitstates. Mit „XPBELL n,n“ können Frequenz und Dauer hingegen variiert werden. Siehe Advanced-BASIC.

BYE

Verwendungszweck:
Verlassen des BASIC-Interpreter zurück in den Debugger-Modus.

Syntax:
BYE <CR>

Wirkungsweise:
Alle gültigen Variablen werden beibehalten und können bei „W“ wie Warmstart später wieder weiter verwendet werden. Wenn Speicherinhalte (SRAM) nicht mit dem Debugger verändert werden, kann ein im Speicher befindliches BASIC-Programm nach „W“ mit „CONT“ an der letzten Zeilennummer fortfahren. Der „Bye“-Befehl kann auch im Programm selbst angewendet werden, um den Debugger während der Laufzeit zu starten.

Beispiel:
10 REM Test
20 PRINT „Hier lauft noch BASIC....“
30 BYE

RUN

Bei Erreichen der Zeile 30 wird der BASIC-Interpreter automatisch verlassen und im Debugger-Mode gestartet.

CLS

Verwendungszweck:
Löschen des aktuellen Bildschirminhaltes (nur Text) bei Verwendung der VGA-Ausgabe mit der SEPIA NET-Karte.

Syntax:
CLS <CR>

Wirkungsweise:
Gesamten Text löschen und anschließend Cursor auf Position links-oben lenken.
Bei erfolgtem CLS wird in der nächsten Zeile die Meldung „Fertig“ quitiert. Der CLS-Befehl gilt nur im Text-Modus anzuwenden. Im VT100-Modus (beispielsweise bei COM-1 Terminal) wird der Befehl „CLR“ verwendet, um den ext. Bildschirm zu löschen. CLR sendet hier eine andere ESC-Sequenz als CLS zum Terminal.

Beispiel:
10 CLS

RUN

Fertig
_

CHR$

Verwendungszweck:
Dieser Befehl wandelt einen nummerischen ASCII-Code in ein entsprechendes Zeichen um.

Syntax:
CHR$ (dezimaler Wert 0...255 oder auch nummerische byte-Variable)

Wirkungsweise:
Das der angegebenen Zahl entsprechende Zeichen nach ASCII kann ausgegeben oder einer Variablen zugewiesen werden. CHR$ wird im allgemeinen verwendet, um ein Sonderzeichen an den Bildschirm oder den Drucker zu schicken. So könnte zum Beispiel bei Einsatz eines COM-Terminals ein CHRS$(7) gesendet werden, um einer Fehlermeldung einen Piepton über den Lautsprecher vorangehen zu lassen, oder es kann ein Formularvorschub CHR$(12) an den Drucker geschickt werden. Eine Beschreibung der Umwandlung von ASCII Zeichen in numerische Zeichen enthält sie ASC-Funktion. ASCII-Codes sind im Anhang von Kapitel 1 dieser Betriebsanleitung aufgelistet.

Beispiele:

PRINT CHR$(66)

Ausgabe: B

Die Umrechung hat die Anzeige des ASCII Zeichencodes 66 zur Folge, der den Großbuchstaben B darstellt.

PRINT CHR$ (13) ; CHR$ (10) ;

Dieser Befehl hat die Anzeige eines Wagenrücklaufzeichens mit anschließendem Form-Feed zur Folge.

Hinweis:
Nicht erlaubt, sind hingeben numerische Ausdrücke >255 oder <0:

PRINT CHR$(256) <CR>
PRINT CHR$(-5) <CR>

Das BASIC meldet in diesem Fehlerfall: „nicht elaubt“

CLEAR

Verwendungszweck:
Dieser Befehl löscht alle Variablen im SRAM-Speicher. In einem Programm muß dieser Befehl zu Beginn stehen, sonst werden nachher unnötige Bytes für Zeichenketten und 182 Bytes für einen #Kanal für Ein- oder Ausgabebetrieb reserviert.

Syntax:
CLEAR[,[Ausdruck1][,Ausdruck2]]

Wirkungsweise:
CLEAR setzt alle numerischen Variablen auf Null, löscht den Inhalt aller Zeichenfolgenvariablen und schließt alle offenen Dateien. Außerdem kann wahlweise die höchste Speichersdresse und der Umfang des für BASIC verfügbaren Speichers festgelegt werden. Das BASIC ordnet Speicherplatz für Zeichenfolgen immer dynamisch zu. Eine „Speicher voll“-Fehlermeldung wird ausgegeben, wenn für BASIC kein freier Speicherbereich mehr verfügbar ist.

Beispiele:

CLEAR <CR>

Alle Variablen und Zeichenfolgen werden gelöscht.

CLEAR 10000 <CR> oder CLEAR N1 (Variable N1 = 1OOOO)

Es werden nun 10000 Bytes für Zeichenketten reserviert. PRINT FRE(X) <CR> ergibt einen Wert, der um 102 Bytes größer, da die Standardreservierung für Zeichenketten entfällt); Print FRE(X$) <CR> ergibt den Wert 10000. Bei zu großen Zahlen erscheint die Fehlermeldung „kein Speicherplatz“, bei Zahlen über 65535 die Meldung „nicht erlaubt“.

Wurde in einem Programm mit dem CLEAR-Befehl zu wenig Speicherplatz für Zeichenketten reserviert, kommt die Fehlermeldung „kein Stringplatz“. Es kann der reservierte Platz auch wieder verkleinert werden. CLEAR 0 beseitigt die Standardreservierung von 100 Bytes für Zeichenketten.

c) CLEAR 1000,2 <CR>

Es werden 1000 Bytes für Zeichenketten und zusätzlich zum Standardkanal noch Speicherplatz für zwei weitere Kanäle für Ein- oder Ausgabebetrieb reserviert. Jeder Kanal braucht weitere 182 Bytes, wie mit FRE(X) beobachtet werden kann.

d) CLEAR,1 <CR>

Es werden alle Variablen gelöscht und für einen weiteren Kanal Speicherplatz reserviert (siehe Warmstart). Bei ausreichendem Speicherplatz könnten theoretisch bis zu 254 Kanäle reserviert werden.

COS

Verwendungszweck:
Befehl zur Umrechung liefert den Cosinus zurück. Es kann eine beliebige positive oder negative Zahl angegeben werden.

Syntax:
COS(x)

Wirkungsweise:
X (Gleitkommawert) muß im Bogenmaß vorliegen. COS ist die trigoniometrische Cosinusfunktion. Zur Umwandlung von Grad in Bogenmaß ist mit PI /180 zu multiplizieren. Die Berechnung erfolgt mit höchster (12 stelliger) Genauigkeit.

Beispiel 1:

```
10  X=2*COS ( . 4 )
20  PRINT  X

RUN
  1.842121988
```

Beispiel 2:

```
10  REM PI = 3.141593
20  PRINT  COS ( PI )
30  GRAD = 180
40  BOGENMASS = GRAD * PI / 180
50  PRINT  COS ( BOGENMASS )

RUN
  -1
  -1
```

CALL

Verwendungszweck:

Aufruf eines in Assemblersprache (Z80 Maschinensprache) erstellen Unterprogramms im SRAM-Speicher. Die CALL-Anweisung empfielt sich für die direkte Verbindung von Assemblerprogrammen mit BASIC. Das Anwender-Unterprogramm hat anschließend bis zu einem RET - Befehl die CPU-Steuerung übernommen.

Syntax:

CALL numerische Variable, [(Argumentenliste)]
CALL Adresse, Wert 1, Wert 2, ..., Wert n

Wirkungsweise:

Die numerische Variable gibt die Startposition des Unterprogramms im SEPIA-Speicher an. Mit diesem Befehl wird zu der angegebenen Adresse in Maschinenspracheroutinen gesprungen. Dabei können mit dem Befehl die angegebenen Werte übertragen werden. Sie werden auf dem Stapel abgelegt und sind so zugänglich. Das BC- bzw. CX-Register enthält die Anzahl n der übertragenen Werte, diese ist nur durch den Speicher begrenzt.Mit RET wird zum BASIC-Programm wieder zurückgesprungen.

Beispiele:

CALL 2024 <CR>

Es wird zur Adresse 2042h gesprungen. Die Adresse kann auch sedezimal angegeben werden, z.B. CALL &7E8.

CALL &8000,45,3 <CR>

Ab Adresse 32768 (8000h) muß sich ein Maschinenprogramm befinden, dem die Werte 45 und 3 übergeben werden. Register BC bzw. CX enthält 00 02, auf dem Stapel befinden sich 03 00 2D 00.

CALL 0 <CR>

Mit CALL 0 wird das BASIC abrupt verlassen und es meldet sich das Boot-Menue vom Betriebssystem. Dieser Zustand entspricht quasi einem „Soft“-RESET ähnlich einem NMI, wobei SRAM-Variablen von BASIC erhalten bleiben. Mit „W“ wie Warmstart kann BASIC mit diesen Speicherinhalten weiter rechnen.

CLOSE

Verwendungszweck:
Schließen von offenen Kanälen wird hier nicht verwendet. Der Befehl beendet ansonsten die Eingabe-/Ausgabeoperation einer Datei, oder einer E/A-Einheit (bsp. Gerät, I/O-Karte).

Syntax:
CLOSE <CR>

CONT

Verwendungszweck
Setzt die BASIC-Programmausführung nach einer Unterbrechung mit STOP oder BREAK fort.

Syntax
CONT

Wirkungsweise:
Wird <CTRL-E> während eines Programmablaufes eingegeben oder kommt der Interpreter zu einem STOP-Befehl in einer Programmzeile, so hält das Programm mit der Meldung „** HALT ** Zeile XYZ“ an, wobei XYZ die Zeilennummer angibt, in der abgebrochen wurde.

Nun kann man mit dem Befehl LVAR die Variablenwerte anschauen, mit LVAR#2 ausdrucken oder die Werte ändern, z.B a = 15 oder a$ = „Text“. Anschließend wird mit dem Befehl CONT das Programm unter Berücksichtigung der eventuell geänderten Werte fortgesetzt. So kann man zu Testzwecken die Variable des Schleifenzählers einer FOR-NEXT-Schleife verändern. Nach einer Programmänderung oder nach Auftreten eines Fehlers ergibt der Befehl CONT die Meldung „CONT geht nicht“. Nach Abbruch mit <CTRL-E> bei einem INPUT-Befehl wird die Eingabeaufforderung wiederholt.

Beispiel:
CONTINUE <CR> oder CONT <CR>

Hinweis:
Der CONT -Befehl eignet sich bestens für die Fehlersuche, da dies dem Benutzer ermöglicht, Unterbrecherpunkte (break points) in Verbindung mit der STOP - Anweisungen zu setzen, Variablen mit Direktanweisungen abzuändern, die Programmausführung fortzuführen, oder mit GOTO die Ausführung bei einer bestimmten Zeilennummer wieder fortzusetzen.

Wurde jedoch eine Programmzeile im BASIC-Code verändert, ist CONT ungültig.

DATA

Verwendungszweck:
Dieser Befehl wird benötigt, um in einem BASIC-Programm beliebige Variablen bestimmte Werte oder Zeichen statisch zuweisen zu können.

Syntax:
DATA *Konstantenliste*
Zahl oder Zeichenkette, durch Komma getrennt (max. 255 Zeichen in einer BASIC-Zeile)

Wirkungsweise:
DATA speichert numerischen Konstanten und Zeichenfolgen, die durch die READ-Anweisung(en) eines Programms angesprochen werden. Arithmetische Ausdrücke sind in der Liste nicht zulässig. Weitere Informationen, siehe RESTORE und READ Anweisungen.

Beispiel 1:
Mögliche Zuweisungen: 10 DATA 1,2,“ HUBER“,MEIER,,,4,ZZ

Sollen Leerzeichen am Anfang einer Zeichenkette mitgelesen werden, so müssen die Zeichenketten in Anführungszeichen gesetzt werden (z.B. „ HUBER“).
(READ-Befehlsform: READ nummerische Variable, Stringvariable, durch Komma getrennt)

```
10 DATA 1,2,“ HUBER“,MEIER,,,4,ZZ
20 READ A,B,A$,B$,C,C$,D(1,1,1),D$(1,3)
30 PRINT A;B;A$;B$;C;C$;D(1,1,1);D4(1,3)
```

Anzeige: 1 2 HUBERMEIER 0 4 ZZ

Die Variable C enthält den Wert 0, C$ ist leer. Der in der READ-Anweisung verwendete Variblentyp (numerisch oder Zeichenfolge) muß mit dem entsprechenden Konstantentyp in der DATA Anweisung immer übereinstimmen.

Beispiel 2:

```
10  PRINT
20  PRINT  „STADT“ , „LAND“ , „PLZ“
30  READ  S$ , L$ , P
40  DATA „Koeln“ , „NRW“, 5000
50  PRINT  S$ , L$ , P

RUN

STADT       LAND        PLZ
Koeln       NRW         5000
```

Dieses kleine Beispiel liest Zeichenfolgen und numerische Daten aus der DATA-Zeile 40.

DIM

Verwendungszweck:
Die DIM -Anweisung spezifiziert die maximalen Werte für Feldvariablenindizes und weist diesen entsprechenden Speicherplatz zu.

Syntax:
DIM Variable(Indizes)[,Variable(Indizes)]

DIM -Befehlsform: DIM Variable 1 (Elementanzahl), Variable 2 (Elementanzahl), usw...

Wirkungsweise:
Zur Reservierung des benötigten Speicherplatzes bei mehr als 10 Elementen dient der DIM-Befehl. Ohne DIM-Befehl am Programmanfang werden pro Matrixvariable 10 Elemente zugelassen (0 - 9). Matrixvariablen (indizierte Variablen) sind Variablen vom Typ A(I), A$(I), A(2,3) oder A$(20,8,2). Doppelte Reservierung für die gleiche Variable ergibt die Fehlermeldung „DIM-Befehl doppelte Zeile xxx“. Die maximale Anzahl von Dimensionen für ein Feld beträgt 255.

Beispiel 1:

```
DIM A$(100),B(20,20),C$(30,30,30),F$(A),D$(X,Y,Z)
```

Die Elementanzahl einer indizierten Variablen kann selber durch eine Variable angegeben werden.

Beispiel 2:

```
10  DIM  A ( 20 )
20  FOR  I=0  TO  20
30  READ  A ( I )
40  NEXT  I
```

In diesem Beispiel werden 21 DATA Anweisungen an beliebiger Stell im Programm gelesen und ihre Werte werden nacheinander A(0) bis einschließlich A(20) zugewiesen. Ist A ein einfach genaues Feld (Standartgenauigkeit), wird Zeile 10 für dieses Feld einen Speicherplatz von 84 Bytes bereitstellen (also 4 Bytes mal 21 Elemente).

Hinweis:
Ein dimensioniertes Feld kann erst innerhalb des Programms neu dimensioniert werden, wenn vorher eine CLEAR Anweisung durchgeführt wurde. Der kleinste Wert für einen Index ist immer 0. Ebenso setzt die DIM-Anweisung alle enthaltenen Elemente der angegebenen Felder auf einen Ausgangswert von Null.

EE

Verwendungszweck:
Das Argument enthält die Eulersche Zahl e, die Basis der natürlichen Logarithmen.

Syntax:
PRINT EE <CR>

Beispiel:
10 X = EE
20 PRINT X

RUN

Anzeige: 2.7182818285

EDIT

Verwendungszweck:
BASIC Quell-Text nachträglich editieren. EDIT zeigt eine bestimmt Zeile an und positioniert die Schreibmarke auf die erste Stelle der Zeilennummer, so daß die Zeile manuell bearbeitet (editiert) werden kann.

Syntax:
EDIT Zeilennummer

oder

EDIT. (Punkt eingeben)

Wirkungsweise:
Um Programmzeilen zu korrigieren oder zu ergänzen, wird der EDIT-Befehl verwendet. Anfangs muß man öfters in der EDIT-Kommandoliste nachschauen. Man kann natürlich bei kurzen Programmzeilen die ganze Zeile neu schreiben, was i.d.R. einfacher geht. Der Editor bearbeitet eine Kopie der Programmzeile in einem Puffer, so dass die eigentliche Programmzeile erst zum Schluß durch die Kopie ersetzt wird. Die Aktuelle BASIC-Zeile ist immer die letzte Zeile, auf die sich eine EDIT Anweisung, ein LIST Befehl, oder eine Fehlermeldung bezog. EDIT ohne Zeilennummer oder Punkt ergibt die Fehlermeldung „nicht erlaubt“.

Beispiel 1:

EDIT 120 <CR>

Es wird die Zeilennummer 120 ausgegeben und auf ein EDIT-Kommando gewartet.

Beispiel 2:

EDIT . <CR> (Punkt eingeben)
Nach einer Fehlermeldung, bei einem Programmabbruch oder erneutem EDIT-Kommando kann die Zeilennummer durch einen Punkt ersetzt werden. Erscheint z.B. nach Eingabe von RUN die Fehlermeldung “Syntaxfehler Zeile 115“, so kann nach Eingabe von EDIT. die Zeile mit den Edit-Kommandos verbessert werden.

EDIT

EDIT-Kommandos:

Wird für n keine Zahl angegeben, gilt n = 1.

nD:	Löscht n Zeichen. Die gelöschten Zeichen werden mit Hochkomma (‘) eingerahmt und angezeigt.
E:	Beendet das Editieren. Die Programmzeile wird durch die Kopie ersetzt, aber nicht angezeigt. Es erfolgt keine „Fertig“-Meldung des Interpreters.
nFX:	Suchbefehl in der Programmzeile: Zeigt die Zeile bis zum n-ten Zeichen X und setzt anschließend den Cursor davor.
H:	Löscht alles rechts vom Cursor und wartet auf Eingabe.
I:	Eingabe von Zeichen ab der Cursorstellung in den Editpuffer und gleichzeitige Anzeige.
nKX:	Löschbefehl in der Programmzeile: Lösche alle Zeichen rechts vom Cursor bis zum n-ten Zeichen X, ohne X zu löschen.
L:	Zeigt die im Puffer enthaltene Programmzeile und setzt den Cursor nach der Zeilennummer an den Anfang der Zeile.
Q:	Abbruch des Editors, die Programmzeile bleibt unverändert und die Meldung „Fertig“ wird ausgegeben.
nR:	Ersetzt n Zeichen ab dem Cursor durch die dann eingebenen Zeichen.
X:	Zeigt die Zeile, bewegt den Cursor an das Zeilenende und wartet auf Zeichen eingabe (I-Modus).
<SPACE>:	Die Leertaste bewegt den Cursor eine Stelle nach rechts, die Zeile wird bis zum Cursor angezeigt.
<DEL>:	bewegt den Pufferzeiger nach links.
<CTRL-E>:	keine Funktion beim EDIT-Befehl.
<ESCAPE>:	beendet den Eingabemodus (I) und bleibt im Editor.
<CR>	beendet den Editor

Mittels Bell-Funktion (CHR$(7)) werden falsche EDIT-Befehle akustisch angezeigt.

ELSE

Verwendungszweck:
ELSE ist Teil der IF-THEN-ELSE oder GOTO Befehlsgruppe.

Syntax:
IF Ausdruck[,]THEN Anweisung(en)[,][ELSE Anweisung(en)]
IF Ausdruck[,]GOTO Zeilennummer[[,]ELSE Anweisung(en)]

Wirkungsweise:
Dei Anweisung „ELSE“ trifft eine Entscheidung hinsichtlich des Programmablaufs, abhängig vom Ergebnis, das ein Ausdruck liefert.

Befehlsform:
IF Bedingung THEN (oder GOTO) Befehlsgruppe 1 ELSE Befehlsgruppe 2

Programmverzweigungen unter bestimmten Bedingungen sind mit dieser Befehlsgruppe möglich, ebenso mehrere aufeinanderfolgende IF-Befehle, die durch logische Operatoren verknüpft sind. Klammern sind bei der folgenden Form erforderlich:

IF X$ = (A$ + B$) THEN ...
sonst erfolgt die Fehlermeldung „Datentypen (Zahl/String) vermischt“.

Beispiel:
```
5  INPUT A, B$
10 IF A = 2 AND B$ = „E“ THEN 30 ELSE 40
20 STOP
30 PRINT “EINGABE RICHTIG“ : END
40 GOTO 10
```

Hinweis:
IF...THEN...ELSE Anweisungen können auch verschachtelt werden. Ein solcher Vorgang ist nur durch die Länge der jeweiligen Zeile eingeschränkt. Das folgende Beispiel stellt eine gültige Anweisung dar:

```
10 IF X>Y THEN PRINT „GROESSER ALS“ ELSE IF Y<X THEN
    PRINT „KLEINER ALS“ ELSE PRINT „GLEICH“
```

Merke:
Ist das Ergebnis von Ausdruck nicht Null (wahr), wird THEN oder GOTO Zeilennummer ausgeführt. Ist das Ergebnis von Ausdruck Null (falsch), wird THEN oder GOTO Zeilennummer ignoriert und die ELSE Anweisung - wenn vorhanden - ausgeführt. Andernfalls wird die Programmausführung mit der nächstmöglichen ausführbaren Anweisung fortgesetzt.
Vor THEN und ELSE kann ein Komma stehen. Auf THEN und ELSE kann entweder eine Zeilennummer folgen, zu der verzweigt werden soll, oder eine oder mehrere auszuführende Anweisungen. Auf GOTO folgt immer eine Zeilennummer.

END

Verwendungszweck:
Beenden der Programmausführung. Der END -Befehl beendet die Programmausführung, schließt alle Dateien und kehrt dann zur BASIC Eingabe-Befehlsebene zurück.

Syntax:
END

Wirkungsweise:
Nach einer solchen Anweisung in einer BASIC-Programmzeile wird das Programm verlassen und die Meldung „Fertig“ angezeigt. Direkt-Befehle wie LIST, EDIT usw. können eingegeben werden. Der Befehl ist nicht unbedingt als Programmende notwendig; er trennt jedoch Unterprogramme, die mit GOSUB aufgerufen werden, vom Hauptprogramm ab. Im Gegensatz zur STOP-Anweisung wird bei der END-Anweisung keine Meldung ausgegeben.

Beispiel:

```
10 IF Z > 1000 THEN END ELSE GOTO 20
20 REM ... weiter ...
```

Wenn der Wert von Z 1000 überschreitet, wird das Programm beendet und es erfolgt eine Rückkehr zur Befehlsebene.

ERL

Verwendungszweck:
ERL übergibt die BASIC-Zeilennummer, in der ein Fehler während der Programm-Laufzeit der Interpreters festgestellt wurde.

Syntax:
n = ERR

Wirkungsweise:
Der in ERL enthaltene, nummerische Wert enthält die Zeilennummer.

Beispiel:
? ERL

Hinweis:
Die Variable ERL enthält die Zeilennummer der Programmzeile, in der der Fehler auftrat. Ein ERL-Fehler bei der Ausführung eines Direktbefehles ergibt den Wert 65535.

ERR

Verwendungszweck:
ERR übergibt einen Fehlercode, welcher in der Fehlertabelle als Nummer für die Ausgabe zugewiesen wurde. Über die Fehlernummer lässt sich der aufgetretene Fehler gem. der BASIC-Fehlertabelle feststellen. Die Fehlertabelle enthält auch reguläre BASIC-Meldungen, wie beispielsweise Nr. 21 mit „BASIC verfügbar“.

Syntax:
n = ERR

Wirkungsweise:
Der in ERR enthaltene, nummerische Wert enthält die zu letzt gespeicherte Fehlernummer.

Beispiel:
? ERR

Hinweis:
Wurde noch kein Fehler erzeugt, ist der Inhalt von ERR = 0.

ERR1

Verwendungszweck:
Wie ERR, jedoch mit angehängter Fehlerspeicher-Nummer bei der Ausgabe.

Syntax:
? ERR1

Wirkungsweise:
Der in ERR enthaltene, nummerische Wert enthält die zu letztgespeicherte Fehlernummer und wird gemeinsam mit der Fehlerspeicher-Nummer erzeugt.

Beispiel:
```
gnampf    : REM irgend ein Text der hier einen Syntaxfehler verursacht
? ERR5
```

Ausgabe:

```
Syntaxfehler
 2  5
```

ERROR

Verwendungszweck:
ERROR simuliert das Auftreten eines Fehlers oder ermöglicht dem Benutzer die Definition von Fehlercodes.

Syntax:
ERROR (Fehlernummer)

Wirkungsweise:
Mit Befehl ERROR XY (0...48) kann die Fehlermeldung XY ausgegeben werden. Der Wert von Ganzzahlausdruck (Fehlernummer) muß immer größer als 0 und kleiner als 255 sein. Fehlermeldungen sind beliebig definierbar, wobei die Texte den Nummern 0...255 zugeordnet werden können. Entsprechend der gewünschten Abbruchbedingung des Programms wird die Fehlernummer angegeben. Die Ausführung einer ERROR -Anweisung, für die keine Fehlerbehandlungsroutine existiert, führt zu einer erneuten Fehlermeldung und dem Abbruch der Programmausführung.

Beispiele:

```
20 IF A > 5 THEN  ERROR 2
```

Wird während des Programmlaufes die Zeile 20 erreicht und der Wert der Variablen A größer 5 ist, dann wird die Zahl 2 als Fehlernummer zwischengespeichert. Zu Beginn des Programms setzt man einen Sprungbefehl:

```
10 ON ERROR GOTO 1000
```

Im Unterprogramm ab Zeile 1000 kann dann der zugehörige Text ausgegeben werden.

```
1000 IF ERR = 50 THEN PRINT „A zu groß“: GOTO ....
```
Nach dem Befehl ON ERROR GOTO ... darf kein CLEAR-Befehl enthalten sein.

Hinweise:
Eine vollständige Liste der bereits in BASIC definierten Fehlercodes und Meldungen finden Sie im vorderen Teil von Kapitel 2 der Betriebsanleitung. Ein vom Benutzer definierter Fehlercode muß einen Wert verwenden, der größer ist als alle Werte, die für SEPIA-BASIC Fehlercodes verwendet werden. Gegenwärtig gibt es 49 BASIC Fehlercodes. Es empfiehlt sich daher, eine Codenummer zu verwenden, die hoch genug ist, wenn weitere Fehlercodes zu hinzugefügt werden sollen.

Fehlerbehandlungsfunktionen

ON ERROR GOTO	Aufruf der Fehlerbehandlung
RESUME	Rückkehr aus Fehlerbehandlungsroutine
ERL	ermittelt Zeilennummer des Fehlers
ERR	ermittelt Fehlercode
ERROR	ermittelt Meldungstext zum Fehlercode

EQV

Verwendungszweck:
Boole´sche Mathematische Funktion für logische Gleichwertigkeit. Abkürzung für Äquivalenz.
(engl. = equivalence)

Syntax:
Ergebnis = Ausdruck1 EQV Ausdruck2

Wirkungsweise:
EQV berechnet (bit-weise) die logische Äquivalenz zweier Ausdrücke.

Ergebnis:
Eine beliebige numerische Variable, die das Ergebnis des Vergleichs aufnimmt.

Ausdruck1, Ausdruck2:
Beliebige Ausdrücke, die Sie vergleichen möchten.

Wenn Sie boolesche Ausdrücke auf ihre Äquivalenz testen, ist das Ergebnis True, wenn die Ausdrücke entweder beide True oder beide False sind. Bei einem bitweisen Vergleich setzt der EQV-Operator das entsprechende Bit im Ergebnis nur dann, wenn das betreffende Bit entweder in beiden Ausdrücken oder in keinem der beiden Ausdrücke gesetzt ist.

Beispiele:

```
10 A = 0
20 B = 0
30 PRINT A EQV B
RUN
Ausgabe:  -1

10 A = 10
20 B = 2
30 PRINT A EQV B
RUN
Ausgabe:  -9
```

Hinweis:

Bit	Vergl.	AND	OR	XOR	EQU	IMP
0 ? 0	=	0	0	0	1	1
0 ? 1	=	0	1	1	0	1
1 ? 0	=	0	1	1	0	0
1 ? 1	=	1	1	0	1	1

EXP

Verwendungszweck:
EXP liefert e (die Basis des natürlichen Logarithmus) hoch x.

Syntax:
EXP(x)

Wirkungsweise:
Bei Überlauf von EXP erscheint eine „Ergebnis zu gross“ BASIC-Fehlermeldung. Der größte, darstellbare Wert mit entsprechendem Vorzeichen wird als Ergebnis übergeben und die Programmausführung wird anschließend fortgesetzt.

Beispiel 1:

PRINT EE^88 <CR> Anzeige: 1.6516362552E+38

Beispiel 2:

```
10  X  =  5
20  PRINT  EXP (X-1)

RUN

 54.59815    ( Anzeige des Wertes e hoch 4 )
```

Hinweis:
x muß kleiner als 88.02969 sein.

EXIT

Verwendungszweck:
FOR ... TO ... STEP ... EXIT ... NEXT
EXIT ist hier Teil einer FOR-NEXT Schleife, um diese bei Erreichen zu verlassen.

Syntax:
EXIT Zeilennummer, Laufvariable

Wirkungsweise:
Mit dem Befehl EXIT wird die Schleife sofort verlassen, bevor die jeweilige Laufvariable ihren Endwert erreicht hat. Dies kann zwar auch mit dem Befehl GOTO durchgeführt werden, jedoch kommt es dann bei nachfolgenden Schleifen zu der Fehlermeldung „NEXT ohne FOR Zeile...“.

Beispiel:
Eine dreidimensionale Matrix wird mit dem Wert 1 aufgefüllt. Mehrere Laufvariablen können, durch ein Komma getrennt, in eine Zeile geschrieben werden.

```
10 FOR I = 1 TO 10
15 INPUT“Eingabe“;X$
20 IF X$ = „e“  THEN 100
25 A$(I) = X$
30 NEXT
40 END
50 :
100 FOR J = 1 TO 10
110 FOR I = 1 TO 10
120 PRINT I;J
130 NEXT I
140 NEXT J
```

Hinweis:
Nach dem Sprung zur Zeile 100 tritt beim ersten NEXT J die Fehlermeldung „NEXT ohne FOR Zeile 140“ auf. Der EXIT-Befehl in einer neuen Zeile 20 beseitigt diesen Fehler:

Ändern Sie Zeile 20 in:

```
20 IF X$  = „e“ THEN EXIT 100,I
```

FOR

Verwendungszweck:
FOR gibt an, wie oft eine Reihe von Instruktionen in einer Schleife durchgeführt weden sollen.

Syntax:
FOR *Variable=x* To *y* [STEP *z*)
..
NEXT [*Variable*] [,*Variable...*]

Wirkungsweise:
Diese Befehle dienen zum Aufbau von Befehlsschleifen.

FOR	Laufvariable = Anfangswert
TO	Endwert
STEP	Schrittweite
...	beliebige Befehle
NEXT	Laufvariable

Beispiele:

```
10 FOR I = -1 TO 1 STEP 0.1
20 PRINT I,“TEST“
30 NEXT I
```

Die Angabe der Laufvariable nach dem Befehl NEXT kann bei nur einer Schleife entfallen. Bei verschachtelten Schleifen muß die jeweilige Laufvariable mit angegeben werden. Solche Schleifen müssen nach dem Schema FOR A..FOR B..FOR C..usw - NEXT C..NEXT B.. NEXT A aufgebaut sein. Da die Rücksprungadressen der Schleifen in einem bestimmten Speicherbereich zwischengespeichert werden, ist auch die Anzahl der verschachtelten Schleifen begrenzt.

```
10 DIM A(15,15,15)
20 FOR I = 1 TO 15
30 FOR K = 1 TO 15
40 FOR L = 1 TO 15
50 A(I,K,L) = 1
60 NEXT L,K,I
```

Hinweise:
Variablen werden als Zähler verwendet. Sie sind numerische Ausdrücke. STEP gibt die Zähler-Schrittweite für jede Schleife an. Der erste numerische Ausdruck ist der Anfangswert des Zählers. Der zweite numerische Ausdruck ist der Endwert des Zählers. Die Programmzeile, die auf FOR-Anweisung erfolgen, werden bis zur NEXT-Anweisung ausgeführt. Darauf hin wird der Zähler um die durch STEP eingestellte Schrittweite weitergezählt.

Wird STEP nicht deklariert, ist die Schrittweite automatisch 1.

FRE

Verwendungszweck:
Mit dem Befehl FRE(X) bzw. FRE(X$) wird der noch freie Speicherplatz im RAM ermittelt.

Syntax:
FRE(*X$*)
FRE(*X*)

Wirkungsweise:
Das numerische Ergebnis enthält die Byte-Zahl der freier SRAM-Adressen. X ist hier nur eine Hilfsvariable und enthält keinen relevanten Wert. Die Argumente (*X$*) und (*X*) sind daher Blindargumente. Bevor FRE(*X$*) den restl. verfügbaren Speicherplatz im zugeordneten Zeichenfolgenspeicher übergibt, leitet BASIC eine interne Defragmentation ein. Die Daten werden im Zeichenfolgenspeicher gesammelt und entsprechend reorganisiert. Ungenutzte Daten-Fragmente werden entfernt und frei gegeben.

Beispiel:
```
10 PRINT FRE(x)

RUN
  31811
```

Hinweis:
Je nach Speicherausbau / Belegung liefert Ihr Rechner möglicherweise ein anderes Ergebnis.

FIX

Verwendungszweck:
Verkürzt *eine Variable* zu einer Ganzzahl.

Syntax:
FIX(X)
FIX$(: Befehlsform: FIX$(Variable , Zeichenanzahl)

Wirkungsweise:
FIX rundet Zahlen nicht auf oder ab, sondern schneidet einfach den Dezimalpunkt und alle Zeichen rechts vom Dezimalpunkt (also auch dahinter) ab.

Beispiel:

PRINT FIX (-58.75)
Anzeige: -58

A = 123.567
PRINT FIX(A)
Anzeige: 123

A = - 0.5
FIX(A)
PRINT FIX(A)
Anzeige: 0

dagegen: PRINT INT(A)
Anzeige: -1

Hinweise:
FIX(*X*) entspricht SGN(*X*)*INT(ABS(*X*)). Der Hauptunterschied zwischen FIX und INT besteht darin, daß FIX nicht die nächstniedrigere Zahl bei negativem *X* liefert. FIX ist ebenso bei der Modulo-Arithmetik sehr nützlich.

FIX$

Verwendungszweck:
String-Operation zum füllen einer Zeichenkette.

Syntax:
FIX$(: Befehlsform: FIX$(Variable , Zeichenanzahl)

Wirkungsweise:
Für Stringoperationen benötigt man oft eine ganz bestimmte Länge einer Zeichenkette.
Dieser Befehl füllt eine vorhandene Zeichenkette mit Leerzeichen (Space) bis zur angegebenen Länge (max. 255 Zeichen) auf, oder schneidet sie ab.

Beispiel :

```
10 INPUT “Wort, Länge“;A$,L
20 IF A$ = „E“ THEN END
30 A$ = FIX$(A$,L)
40 PRINT A$;“Länge“;LEN(A$)
50 GOTO 10
```

FIND

Verwendungszweck:
Suchen von Zeichen bzw. Zeichenketten

Syntax:
FIND Trennzeichen Zeichenkette Trennzeichen , Zeilenbereich

Wirkungsweise:
Beliebige Zeichenketten, z.B. Variablennamen können in einem Programm ermittelt werden. Es werden die Zeilen mit der gesuchten Zeichenkette ausgegeben.

Als Trennzeichen können beliebige ASCII-Zeichen im Bereich von 21h bis 40h verwendet werden mit einigen Ausnahmen: keine Zahlen, kein Komma. Wird ein als Trennzeichen verwendbares Zeichen gesucht, darf dieses natürlich nicht als Trennzeichen verwendet werden. FIND ... ergibt die Fehlermeldung „Syntaxfehler“ und muß als FIND ‘.’ eingegeben werden.

Beispiel:

FIND „DATA“,100-200 <CR>

Es wird im Programm von Zeile 100 bis Zeile 200 nach der Zeichenkette DATA gesucht. Dabei ist zu beachten, dass BASIC-Befehle nach der Eingabe in Großbuchstaben umgesetzt sind und in Kleinbuchstaben nicht gefunden werden (FIND ‘gosub’ findet nichts).

FIND .Eingabe. <CR>

Es wird das Wort Eingabe im ganzen Programm gesucht.

GOTO

Verwendungszweck:
Sprungbefehl zu einer neuen BASIC-Zeile. Unbedingte Vezweigung aus dem normalen Programmablauf zu einer bestimmten Zeilennummer im BASIC-Programm.

Syntax:
GOTO Zeilennummer

Wirkungsweise:
Der Befehl kann wie RUN zum Start eines Programmes verwendet werden, wenn die Zeilennummer der ersten BASIC-Zeile angegeben wird. Variableninhalte werden dann nicht gelöscht. Es kann auch direkt zu einer beliebigen Zeilennummer gesprungen werden, um dort das Programm fortzusetzen. Existiert die angegebene Zeilennummer nicht, ist die Folge eine Fehlermeldung: „Sprungziel fehlt“.

Beispiel:

```
100 GOTO 200
150 STOP
200 PRINT „Zeile 200“
```

Der STOP-Befehl in Zeile 150 wird übergangen, da von Zeile 100 direkt nach Zeile 200 gesprungen wird.

Hinweis:

IF Bedingung THEN GOTO ...

Programmverzweigungen (Then GOTO) unter bestimmten Bedingungen (If) sind mit dieser Befehlsgruppe ebenso möglich.

```
100 A =   5
110 IF A = 5 Then GOTO 200
150 STOP
200 PRINT „Zeile 200“
```

GOTO kann ebenso in Kombination mit ON ERROR verwendet werden:

```
10 ON ERROR GOTO 1000
1000 STOP
```

Im Unterprogramm ab Zeile 1000 kann dann STOP oder der zugehörige Text ausgegeben werden.

GOSUB

Verwendungszweck:
Der Interpreter springt zur angegebenen Zeilenummer und setzt dort seine Arbeit fort, bis der Befehl RETURN auftaucht. Dabei wird die Rücksprungadresse gespeichert. Mehrere GOSUB-Befehle können nacheinander folgen.

Syntax:
GOSUB Zeilennummer ... RETURN

Wirkungsweise:
Springt zur angegebenen Zeilennummer einer Sub-Routine, und kehrt mit Return zurück.

Beispiel:

```
100 A = 4
110 IF A = 4 Then GOSUB 1000
150 STOP
1000 PRINT „A = 4“
1100 RETURN
```

Andere Beispiele mit GOSUB:

ON Z GOSUB Zeilennummer A, Zeilennummer B, usw.

Entsprechend der Zahl Z (0 - 255) wird zu den angegebenen Zeilen verzweigt, bei Z = 2 also zur Zeilennummer B. Ist Z = 0 oder größer als die Anzahl der angegebenen Zeilennummern, so wird der nächste Befehl nach dem vollständigen GOSUB-Befehl bearbeitet.

RETURN: Befehlsform: RETURN Zeilennummer

Nach der Bearbeitung eines mit dem Befehl GOSUB erreichten Unterprogrammes wird nicht hinter diesen Befehl gesprungen, sondern zu der angegebenen Zeilennummer. So kann über GOSUB-Ebenen hinweg gesprungen werden.

ON Z RETURN Zeilennummer A, Zeilennummer B, usw.

Wie beim GOSUB-Befehl kann auch unter bestimmten Bedingungen der RETURN-Befehl ausgeführt werden.

Hinweise:
Ein Unterprogramm kann in einem Programm beliebig oft aufgerufen werden. Unterprogramme lassen sich ebenso von anderen Unterprogrammen aus aufrufen. Die Verschachtelung von Unterprogrammen ist nur durch den verfügbaren RAM-Speicherplatz begrenzt. Unterprogramme können an beliebiger Stelle des BASIC-Programms stehen. Der bessern Übersicht halber sollten Unterprogramme jedoch deutlich vom Hauptprogramm zu unterscheiden sein.

HEX$

Verwendungszweck:
Diese Stringfunktion wandelt eine als Argument angegebene Dezimalzahl (abgerundeter Wert) im Bereich von +65535 bis -65535 in eine Zeichenfolge als String um, welcher die Sedezimalzahl in hexadezimaler Schreibweise enthält.

Syntax:
X$ = HEX$(Zahl)

Wirkungsweise:
Bei Hex-Zahlen muss das &-Zeichen bei sedezimaler Schreibweise immer vorangestellt werden. Die HEX$-Funktion setzt 16-Bit breite Dezimalwerte in eine hexadezimale Zeichenfolge im Bereich von 0000 bis FFFF als String um.

Beispiele:

```
PRINT HEX$(1000)                <CR>      Anzeige    : 03E8
A = 2020 <CR> PRINT HEX$(A) <CR>          „          : 07E4
PRINT HEX$(18.9)                <CR>      „          : 0012
PRINT (&3E8)                    <CR>      „          : 1000

10 INPUT “HEX-ZAHL“;A$          : REM Eingabe einer sedezimalen Zahl
20 PRINT VAL(„&“+A$)            : REM mit Großbuchstaben, Ausgabe der Dezimalzahl
```

Hinweise:
Sedezimale Zahlen sind Zahlen mit Basis 16, im Gegensatz zu Dezimalzahlen mit Basis 10. Wenn der Wert negativ ist, wird das Zweier-Komplement (binär) verwendet.

Rechengenauigkeit des Interpreters:
Die folgenden Beispiele zeigen, dass dieser BASIC-Interpreter weniger Probleme mit Rundungsfehlern hat als andere.

```
X = 66/20 <CR> PRINT INT(10*X)/10   <CR> Anzeige: 3.3
X = SQR(2) <CR> PRINT X^2           <CR> Anzeige: 2
PRINT LOG(0.99995)                  <CR> Anzeige: -5.0001249077E-05
```

Allerdings kann aufgrund der binären Zahlendarstellung bei einer Schleife von -1 nach +1 in Schritten 0.1 der Wert Null nie erreicht werden.

Beispiel:

```
10 FOR  N = -1 TO 1 STEP .1       Nach der Anzeige  der  Werte
20 PRINT N                        -1, -.9 bis -.2 kommen
30 IF N=0 THEN PRINT „N = 0“      -.099999999999  und
40 NEXT                           1.1368683772E-12
```

INP

Verwendungszweck:
Empfängt ein I/O-Byte vom Eingabeanschluß (Hardware-Port) des Rechners. Von BASIC aus werden mit diesem Befehl die Eingangs-Ports direkt angesprochen, wobei ein Byte mit 8-bit Breite gelesen und dezimal angezeigt wird. Die Ausgabe eines Bytes erfolgt mit dem Befehl OUT. Somit ist INP die Komplementärfunktion der OUT Anweisung.

Syntax:
INP(Port n)

Wirkungsweise:
n ist eine gültige Anschlußnummer (Portnummer) im Bereich von 0 bis 65535. Bei der hier verwendeten Z80-CPU sind Portnummern größer 255 jedoch nicht erlaubt. 16-bit I/O-Zugriffe sind i.d.R. nur über 16-bit Prozessoren, spezielle Hardware, oder mittels spezieller Advanced-BASIC-Befehle möglich. Port = 256 ergibt die Fehlermeldung „nicht erlaubt“.

Beispiel:
PRINT INP(&10) <CR>

Der Wert kann auch einer Variablen zugewiesen werden:

```
10 A = INP(&10)
20 PRINT A
```

Hinweis:
Bei Ausführung enthält die Variable A den am Anschluß 10h bereitstehenden Dateninhalt. Der gelieferte Wert ist 8-bit breit (Dezimalwert im Bereich 0 bis 255).

INPUT

Verwendungszweck:
INPUT erwartet ein Zeichen über die Tastatureingabe und bereitet das Programm auf diese Eingabe während der Programmausführung vor.

Syntax:
INPUT „beliebiger Text“ ; Variable 1, Variable 2, usw...

Wirkungsweise:
Mit diesem Eingabe-Befehl werden Zahlen in nummerische (A0 - Z9) und Zeichenketten in Stringvariablen (A0$ - Z9$) eingelesen (siehe VARPTR). Variablenkennzeichnungen mit mehr als zwei Buchstaben sind erlaubt, werden aber nicht unterschieden. <CR> als Eingabe führt nicht zum Programmabbruch, sondern übernimmt den Wert 0 bei nummerischen bzw. einen Leerstring bei Stringvariablen (siehe auch: INPUT#).

Beispiele:

```
10 INPUT „EINGABE“;A
```

Auf dem Bildschirm erscheint nach dem Wort „Eingabe“ ein Fragezeichen (siehe OPTION) und der Interpreter wartet solange, bis eine Zahl über die Tastatur eingegeben wurde. Der Versuch, Buchstaben einzugeben, ergibt hier die Fehlermeldung „* Falsche Eingabe *“. Sedezimale Zahlen werden immer mit vorangestelltem „&“-Zeichen und mit Großbuchstaben eingegeben.

```
10 INPUT „Eingabe“;A1,BA$,A2
```

Nach der Ausgabe des Fragezeichens können eine Zahl, eine Zeichenkette und wieder eine Zahl, durch Komma getrennt, eingegeben werden. Wird die erste Eingabe mit <CR> abgeschlossen, so erscheinen zwei Fragezeichen, ohne dass der Text in Anführungszeichen nochmal ausgegeben wird. Bei der zweiten Eingabe (Variable BA$) ergibt ein Komma die Fehlermeldung „* Komma falsch *“. Zeichen nach dem Komma werden nicht übernommen.

Hinweis:
Die eingegebene Daten werden den angegebenen Variablen zugewiesen. Die Anzahl der übergebenen Daten muß mit der Anzahl der Variablen zwingend übereinstimmen.

IF

Verwendungszweck:
IF ist ein Entscheidungsbefehl welcher Bedingungen mit einander vergleicht. IF verändert entsprechend den weiteren Programmablauf, abhängig vom Ergebnis (wahr, oder nicht wahr).

Syntax:
IF (true) THEN ... ELSE ... GOTO ...
Befehlsform: IF Bedingung THEN (oder GOTO) Befehlsgruppe 1 ELSE Befehlsgruppe 2

Wirkungsweise:
Mit der IF - THEN - ELSE Anweisung können mehrere Bedingungen formuliert werden.
Ist ein Ergebins wahr (IF nummerischer Ausdruck nicht NULL), ist die Bedingung erfüllt, so dass THEN weitere Zuweisungen vornehmen, oder zu anderen Programmabschnitten mit GOTO springen kann. Bei Nichterfüllung kann mit ELSE weiter verzweigt werden.

Beispiel:

IF...THEN (GOTO)...ELSE

Programmverzweigungen unter bestimmten Bedingungen sind mit dieser Befehlsgruppe möglich, ebenso mehrere aufeinander folgende IF-Befehle, die durch logische Operatoren verknüpft sind.

Die Klammern sind bei der folgenden Frageform erforderlich: IF X$ = (A$ + B$) THEN .. , sonst erfolgt die Fehlermeldung „Datentypen (Zahl/String) vermischt“.

```
10 INPUT A,B$
20 IF A = 2 AND B$ = „E“ THEN 30 ELSE 40
25 STOP
30 PRINT “EINGABE RICHTIG“ : END
40 GOTO 10
```

Hinweis:
Einfach „übersetzt“ kann man IF - THEN - ELSE auch mit folgenden Worten darstellen:

If	=	WENN	die nachfolgende Bedingung erfüllt ist
Then	=	DANN	führe die nachfolgende Anweisung / Sprung aus
Elese	=	SONST	arbeite mit einer anderen Anweisung / Sprung weiter

Grundsätzlich kann für die Abfrage vor und hinter dem Vergleichsoperator entweder eine Variable, eine Konstante, oder ein Rechenausdruck stehen. Nach dem Befehlswort „THEN“ können auch mehrere Anweisungen stehen, welche die Routine ausführt, wenn das Ergebnis zuvor als „wahr“ interpretiert wurde. Bei Nichterfüllung der IF-Bedingung (Operantenvergleich ist unwahr) wird die IF-Zeile im Programm übersprungen, wenn kein ELSE verwendet wird.

INKEY$

Verwendungszweck:
Zeichenkettenvariable (String), welche eine Eingabe an der Tastatur erwartet.

Syntax:
A$ = INKEY$

Wirkungsweise:
Der Befehl wartet auf eine Tasteneingabe (nur ein Zeichen) und weist einer String-Variablen das Zeichen aus dem Tastaturpuffer zu.

Beispiel:
```
10 A$ = INKEY$
20 PRINT A$
```

INSTR

Verwendungszweck:
Suchen der Position von Zeichen in Strings.

Syntax:
INSTR („Zeichenkette 1“, „Zeichenkette 2“, Startzeichen, Zeichenanzahl)

Wirkungsweise:
Dieser Befehl vergleicht den Inhalt von Zeichen und durchsucht in einer Zeichenkette (Stringvariable oder Zeichenkette in Anführungszeichen) nach einer zweiten Zeichenkette. Die Suche beginnt ab dem ersten Zeichen, oder falls angegeben, ab dem Startzeichen und erstreckt sich über soviele Zeichen wie mit der Zeichenanzahl angegeben sind.

Beispiele:

```
10 PRINT INSTR(„aabbccddeeffgg“,“bb“)      <CR>     Anzeige  : 3
20 PRINT INSTR(„aabbccddeeffgg“,“hh“)      <CR>        „     : 0
30 PRINT INSTR(„abcdefabcdefgh“,“cd“,4)    <CR>        „     : 9
```

In Zeile 20 ist das Ergebnis NULL, da keine „hh“ gefunden wurden.

- - - - -

```
10 A$ = „ABCDEB“
20 B$ = „B“
30 PRINT INSTR(A$,B$); INSTR(4,A$,B$)

RUN
Anzeige:  2  6
```

In diesem einfachen Beispiel wird das Zeichen „B“ mit dem Inhalt der Zeichenkette A$ hier „ABCDB“ durchsucht. Wird die Zeichenkette von Anfang an verglichen, dann wird das Zeichen „B“ an der Text-Position 2 gefunden. Wird hingegen erst ab Position 4 mit der Suche begonnen, dann wird „B“ erst an der Text-Position 6 identifiziert.

INT

Verwendungszweck:
Bilden einer Integer-Zahl als größte oder ganze Zahl von x ohne Nachkommastellen.

Syntax:
z = INT(x)

Wirkungsweise:
Mit INT kann eine Gleitkommazahl in eine ganzzahlige Zahl umgewandelt werden. Dabei wird entsprechend der Nachkommastellen auf- oder abgerundet. Doch Achtung: Positive Zahlen werden immer abgerundet, negative, werden in negativer Richtung aufgerundet.

Beispiel:

PRINT INT(45.67)
Anzeige: 45

PRINT INT(-2.89)
Anzeige: -3

IMP

Verwendungszweck:
Boole´scher (bit-weiser) Operator für logische Ungleichheit (Implizierung).

Syntax:
v = A IMP B

Wirkungsweise:
IMP wird benutzt, um logische Folgerungen auszuwerten. Die IMP-Bedingung gilt dann als erfüllt, wenn: Wert1 (A) nicht Falsch ist, und Wert2 (B) ist nicht Wahr ist. Das Ergebnis ist Wahr (true), wenn der Wert -1 zurückgegeben wird.

Beispiele:

PRINT 1 IMP 1 <CR>
Anzeige: -1

PRINT 2 IMP 2 <CR>
Anzeige: -1

PRINT 1 IMP 2 <CR>
Anzeige: -2

PRINT 2 IMP 1 <CR>
Anzeige: -3

Hinweis:
In Klammen gesetzte Operationen haben immer Vorrang. Innerhalb der Klammer gilt jedoch die normale Rangfolge der Operatoren. Bei Operatoren in einer Zeile nimmt die Priorität von links nach rechts ab. Alle logische Operationen setzen ihre Operatoren intern in eine 16-Bit Ganzzahl um. Dabei ist der Bereich von 0..65535 (FFFFh) oder -32768 bis 32767 erlaubt (sonst erfolgt Fehlermeldung: nicht erlaubt).

Wahrheitstabelle:

```
A     B          A IMP B
-----------------------
T     T      =       T
T     F      =       F
F     T      =       T
F     F      =       T

(T = True, F=False)
```

KILL

Verwendungszweck:
Loeschung von DIM-reservierten Variablen/Bereichen.

Syntax:
KILL X, Y, ... usw ...

Wirkungsweise:
Variablen-Bereiche, welche zuvor mit dem DIM-Befehl zugewiesen wurden, können mit dem KILL-Befehl explizit gelöscht werden. CLEAR hingegen löscht alle Variablen.

Beispiel1:
```
10 DIM A$ 10
20 DIM B$ 10
30 DIM C$ 10
40 A$ = „Nur “
50 B$ = „ein “
60 C$ = „Test  “
70 KILL B$
80 PRINT A$; B$; C$
```

RUN <CR>

Ausgabe: Nur Test

Befehlsform: KILL erste Variable, zweite Variable, ...

Sind im Programm für größere Variablenmengen mit dem DIM-Befehl entsprechende Speicherbereiche reserviert, so kann die Reservierung wieder einzeln aufgehoben werden. Das Dollar-Zeichen kennzeichnet hier Variablen für Zeichenketten.

Es werden die Dimensionierungen von A$, B$ und C$ aufgehoben. Die Löschung einer nicht existierenden Dimensionierung ergibt keinen Fehler, da keine weitere Prüfung erfolgt.

Beispiel2:
KILL A, B <CR>

Die Dimensionierung für Zahlen in Variable A und B werden gelöscht.

Hinweis:
Vorsicht: Verwechselungsgefahr! Der „KILL“-Befehl wird in anderen BASIC-Varianten eventuell als Datei-Lösch-Befehl auf Datenträger verwendet. SEPIA benutzt hierzu jedoch nicht den KILL-, sondern den ERASE-Befehl. Siehe Advanced-BASIC.

LET

Verwendungszweck:
LET veranlasst eine Zuweisung von Werten an eine Variable.

Syntax:
LET K = (x)

Wirkungsweise:
Die LET-Anweisung erlaubt einen Festwert (als nummerischen Ausdruck) einer zu bildenden Variablen zuzuschreiben. Durch die LET-Anweisung können Werte für jedes Abfrageergebnis auch neu berechnet werden.

Beispiel:

```
10 LET K = 1.2345678
20 N = K + 1
30 PRINT K, N
```

Bei SEPIA gehts auch ohne:

```
10 K = 1.2345678
20 N = K + 1
30 PRINT K, N
```

Hinweis:
In der Regel wird aus Gründen der Übersichtlichkeit die Variable im Programm anschließend als Konstante verwendet. Die Zuweisung kann ebenso direkt (also auch ohne LET) erfolgen. In älteren BASIC-Interpretern und/oder Beispielprogrammen wird jedoch oft noch die LET-Anweisung verwendet.

LVAR

Verwendungszweck:
LVAR unterstützt die Ausgabe der BASIC-Variableninhalte als Listing.

Syntax:
LVAR

Wirkungsweise:
Die in einem Programm verwendeten Variablen und ihre Werte können auf den Bildschirm (keine Kanalangabe oder #0), bzw. auf den Drucker (Kanal #2), ausgegeben werden, jedoch erst, nachdem das Programm mit LOADGO geladen und gestartet, oder manuell mit RUN gestartet wurde, da bei diesem Vorgang erst eine Zuordnung erfolgt.

Beispiele:

LVAR <CR>
Ausgabe der verwendeten Variablen am Bildschirm.

LVAR#2 <CR>
Ausgabe auf den Drucker

Hinweis:
Die Ausgabe kann wie beim LIST-Befehl unterbrochen bzw. abgebrochen werden (siehe LIST-Befehl). Es werden keine dimensionierten Variablen angezeigt.

LIST

Verwendungszweck:
Es werden die programmierten BASIC-Programmzeilen auf den Bildschirm oder Drucker ausgegeben. Ohne Kanal-Angabe erfolgt die Ausgabe am Bildschirm, mit LIST#2 über den Kanal Nr.2 auf den LPT1-Drucker.

Syntax:
LIST <CR>

Wirkungsweise:
Mit dem Befehl OPTION kann die vorgegebene Zeilenlänge von 80 Zeichen getrennt für Bildschirm und Drucker verändert werden (siehe OPTION). Eine Zeilennummer kann durch einen Punkt ersetzt werden. Die Leertaste <SPACE> stoppt die Ausgabe, mit nochmaligem Drücken wird das Listing fortgesetzt, mit <CTRL-E> abgebrochen (Meldung: * Abbruch *).

Beispiele:

LIST <CR>
Das Programm wird von Anfang an am Bildschirm aufgezeigt.

LIST 100 <CR>
Die Programmzeile 100 wird angezeigt. LIST 0 ist möglich, wenn die Zeile 0 vorhanden ist.

LIST 130-250 <CR>
Ausgabe ab Zeilennummer 130 bis 250 einschließlich.

LIST -200 <CR>
Ausgabe ab der ersten Zeile bis zur Zeile 200.

LIST 1000- <CR>
Ausgabe ab Zeile 1000 bis zum Programmende.

LIST . -200 <CR> oder LIST.- <CR>
Ausgabe ab der zuletzt bearbeiteten Zeile bis Zeile 200 oder bis zum Programmende.

Hinweis:
Bei der Programmierung dürfen nur Zeilennummern von 0 bis 65529 verwendet werden.
Die Ausgabe mit LIST#2 entspricht weitläufig dem LLIST-Befehl.

LOG

Verwendungszweck:
LOG berechnet den natürlichen Logarithmus eines Arguments, das größer ist als NULL.

Syntax:
(v) = LOG(x)
LOG(: Es wird der natürliche Logarithmus (Logarithmus zur Basis e) berechnet.

Wirkungsweise:
Der natürliche Logarithmus ist nur für positive Zahlen verwendbar. Bei x = 0 (NULL) oder negativ, wird eine Fehlermeldung „nicht erlaubt“ ausgegeben. Daher sollte erst immer eine Plausibilitätskontolle vorausgehen, damit (x) nie <= 0 ist.

Beispiel:

PRINT LOG(1000) <CR> Anzeige: 6.907755279

PRINT LOG(EE) <CR> Anzeige: 1

Hinweis:
LOG ist die komplimentäre Umkehrfunktion zu EXP.

LOG10

Verwendungszweck:
LOG10 berechnet den natürlichen Logarithmus eines Arguments, das größer ist als NULL, zur Basis 10.

Syntax:
(v) = LOG10(x)
LOG10(: Es erfolgt die Berechnung des dekadischen Logarithmus (zur Basis 10).

Wirkungsweise:
Der dekadische Logarithmus ist nur für positive Zahlen verwendbar. Bei x = 0 (NULL) oder negativ, wird eine Fehlermeldung „nicht erlaubt“ ausgegeben. Daher sollte erst immer eine Plausibilitätskontolle vorausgehen, damit (x) nie <= 0 ist.

Beispiel:

PRINT LOG10(1000)	<CR>	Anzeige: 3
PRINT LOG10(EE)	<CR>	Anzeige: .4342944819

LEN

Verwendungszweck:
Numerische Länge einer Zeichenkette (String) ermitteln.

Syntax:
(y) = LEN(Zeichenkette)

Wirkungsweise:
LEN liefert die Anzahl der Zeichen, die in einer String-Zeichenkette vorhanden sind.

Beispiel:

```
10 B$ = „ABCDEF“
20 L = LEN(B$)
30 PRINT L

RUN
 6
```

Hinweis:
Da Strings nie mehr als 255 Zeichen enthalten, liegt das Ergebnis zwischen 0...und 255.
Leer- und Sonderzeichen werden mitgezählt.

LEFT$

Verwendungszweck:
Diese Funktion übergibt n Zeichen einer Zeichenkette, von links an.

Syntax:
LEFT$ (Zeichenkette,n)

Wirkungsweise:
Kopiert einen Teil-String in einen neuen String mit n Zeichen.

Beispiel:
```
10 A$ =“ABC123“
20 B$ = LEFT$(A$,4)
30 PRINT B$

RUN
 ABC1
```

Hinweis:
Siehe auch MID$ und RIGHT$.

LOCATE

Verwendungszweck:
Diese Anweisung dient zur direkten X-Y-Positionierung von Zeichen auf dem Bildschirm.

Syntax:
LOCATE x,y

Wirkungsweise:
Das auszugebende Zeichen (oder String) wird an eine bestimmte Stelle ausgegeben. Die Koordinate X entspricht der Spalte, Y entspricht der Zeile, an der die Zeichenfolge erscheinen soll. Dabei müssen die Grenzen der möglichen Koordinaten eingehalten werden. Sämtliche Angaben erfolgen als Ganzzahl.

Beispiel:

```
10 CLS
20 LOCATE 10,12
30 PRINT „Hallo“
```

Die Ausgabe „Hallo“ erscheint in Spalte 10 Zeile 12.

Hinweise:
Der Ursprung, auf den sich diese XY-Koordinaten beziehen, liegt oben-links an Position 1,1. Da die VGA-Karte von SEPIA 80 Zeichen in 35(36) Zeilen darstellt, düfen diese Koordinaten nicht überschritten werden. Werden dennoch höhere Koordinaten verwendet, scrollt der Bildschirm entsprechend hoch, sodass die Zeichensausgabe stattfinden kann. Dabei können zuvor erzeugte Textdarstellungen in den oberen Zeilen gelöscht werden (nach oben rutschen). Die Koordinate 0,0 gibt es nicht und wird wie 80,36 behandelt.

Merke:

Min. Spalten	(X) = 1
Min. Zeile	(Y) = 1
Max. Spalten	(X) = 80
Max. Zeile	(Y) = 35

MID$

Verwendungszweck:
Austausch von String-Teilen. Diese Funktion ersetzt Zeichen einer Zeichenkette, mit n Zeichen einer anderen Zeichenkette.

Syntax:
x$ = MID$(„Zeichenkette“ oder Stringvariable, Startzeichen, Zeichenanzahl)

Wirkungsweise:
Kopiert einen Teil-String an eine bestimmte Stelle im Haupt-String. Eine durch das Startzeichen und die Zeichenanzahl bestimmte Folge von Zeichen wird von links beginnend in der Zeichenkette gesucht und übernommen.

Beispiel:

```
10 INPUT"Wort"; A$
20 IF A$ = „E" THEN END
30 INPUT "Startzeichen";SZ
40 INPUT "Zeichenanzahl";ZA
50 B$ = MID$ (A$,SZ,ZA)
60 PRINT B$ : GOTO 10
```

Eingabe Wort:	Startzeichen:	Zeichenanzahl:	Anzeige:
Testwort	1	3	Tes
„	5	1	w
„	0	3	nicht erlaubt
„	1	10	Testwort
„	<CR>	<CR>	nicht erlaubt

Hinweis:
In einem vorhandenen String kann ab einem Startzeichen eine bestimmte Zeichenanzahl mit Zeichen aus einer Zeichenkette überschrieben werden. Dies ist wesentlich schneller als eine Teilung und Ergänzung einer Zeichenkette.

Siehe auch LEFT$ und RIGHT$.

MOD

Verwendungszweck:
Mathematische Funktion Modulus. Sie gibt den Rest einer Division zurück.

Syntax:
MOD

Wirkungsweise:
Mit MOD wird der verbleibende Zahlenrest einer Ganzzahl-Division zurückgeggeben.

Beispiel:

Es wird der Rest ermittelt, den man bei der Division des Dividenden durch den Divisor erhält.

PRINT 9 MOD 2 <CR> Anzeige: 1

PRINT 8 MOD 2 <CR> Anzeige: 0

NEW

Verwendungszweck:
Löscht alle Variablen und startet den BASIC-Interpreter neu.

Syntax:
NEW

Wirkungsweise:
Das aktuelle Programm wird im Speicher gelöscht. Alle Variablen werden zurück gesetzt. BASIC wird neu initialisiert. Es erfolgt erneut die Startmeldung vom BASIC-Interpreter.

Beispiel:
NEW <CR>

NEXT

Verwendungszweck:
Siehe FOR ... TO ... STEP ... EXIT ... NEXT

Diese Befehle dienen zum Aufbau von Befehlsschleifen.

Befehlsform:

FOR	Laufvariable = Anfangswert
TO	Endwert
STEP	Schrittweite
...	beliebige Befehle
NEXT	Laufvariable

NOT

Verwendungszweck:
Logischer, bit-weiser Operator.

Syntax:
IF x NOT 1 THEN...

Wirkungsweise:
Berechnet binäre Negation für einen Boolean-Ausdruck.

Beispiel:
10 A = 1
20 ? NOT A

RUN
-2

Hinweis:
Logische Operatoren wie NOT, AND, OR, XOR, EQV und IMP, werden im Zusammenhang mit Vergleichsoperationen verwendet. Oft werden logische Operatoren bit-weise genutzt. So kann die bit-Antwort bei einem Vergleich „Wahr“ (-1) oder „Falsch“ (0) ergeben, damit sie als boole´sche Anweisung in Programmen Verwendung findet. Bei bit- weiser Verwendung können so schnelle Entscheidungen zwischen zwei Möglichkeiten getroffen werden. Sie dienen quasi als schnelle Schalter für weitere Anweisungen.

Wahrheitstabelle:

```
A         NOT
-------------
T     =     F
F     =     T

(T = True, F=False)
```

OUT

Verwendungszweck:
Gegenstück zu INP Befehl. Mit OUT wird ein Hardware-Port über eine anzugebene (8/16-bit) I/O-Adresse byte-weise geschrieben.

Syntax:
OUT(adr),byte

Wirkungsweise:
Direkter Port-Zugriff auf die Hardware. Sendet ein Byte auf ein Ausgabe-Register.

Beispiel 1:

OUT (&F2),123 <CR> Schreibt 123 auf I/O-Adresse F2h.

Beispiel 2:

Es geht auch Dezimal, über eine Variable:

```
10 ADR = (242)
20 OUT(ADR),123

RUN
```

Hinweis:
Bei OUT auf Z80-Systemen sind Adresse und Byte jeweis 8-bit breit (0...255). Bei Nutzung der OUT-Befehle muss man jedoch sehr vorsichtig sein, da jedes System bereits Adressen für die eigene Hardware (bsp. COM-Port, LPT-Port, Interrupt-Register, RTC...) nutzt und somit nicht alle Adressen in diesem Rahmen frei verfügbar sind.

Anmerkung:
Seriöse Hersteller geben dem Nutzer Schaltpläne und eine Adress-MAP an die Hand, aus der die gesamte Adressbelegung sämtlicher I/O-Bauteile ersichtlich ist.

OUTBYTE

Verwendungszweck:
Ausgabe eines Zeichen auf den genannten Kanal.

Syntax:
OUTBYTE #kanal,byte

Wirkungsweise:
Kanal 0 ist die Hauptkonsole, sodass bei Angabe von OUTBYTE 0,67 direkt ein Byte auf COM1 ausgegeben wird. 67 hat die ASCII-Nummer des Großbuchstaben „C“. Das folgende Besipiel gibt nach <CR> entsprechend C aus. Als Byte sind die Zahlen 0...255 erlaubt.

Beispiel:
OUTBYTE #0,67

Ausgabe:
C

Unterschied von OUTBYTE zum Print-Befehl:

```
1 REM PRINT
5 A = 1
10 FOR I = 1 TO 10
20 PRINT A;              Ausgabe: 1 1 1 1 1 1 1 1 1 1
30 NEXT
```

dagegen:

```
1 REM OUTBYTE
5 A = 49
10 FOR I = 1 TO 10
20 OUTBYTE#0,A           Ausgabe: 1111111111
30 NEXT
```

ON

Verwendungszweck:
Anweisung dient als Sprungziel mit Zeilennumerrangabe oder zur Fehlerbehandlung.

Syntax:
ON ERROR GOTO....
Fehlerbehandlung (ON ERROR GOTO, ERR und ERL).

ON X GOTO ...
Sprungbehandlung, wenn X

Wirkungsweise:
Verzweigt den nachfolgenden Befehl weiter zur angegebenen Zeilennummer im Programm. Das Argument muss als Ganzzahl größer 0 und kleiner als 256 sein.

Beispiele:

```
10 ON ERROR GOTO 100
20 REM ...
100 PRINT „Fehler gefunden“
```

```
10 X = 1
20 ON X GOTO 100
100 PRINT „Alles O.K.“
```

- - - - -

ON Z GOTO Zeilennummer A, Zeilennummer B, usw.

Die Wirkungsweise ist dieselbe wie beim GOSUB-Befehl, ein Rücksprung kann jedoch nicht erfolgen.

ON Z GOSUB Zeilennummer A, Zeilennummer B, usw.

ON Z RETURN Zeilennummer A, Zeilennummer B, usw.

Wie beim GOSUB-Befehl kann auch unter bestimmten Bedingungen der RETURN-Befehl ausgeführt werden.

Hinweis:
Nach Aktivierung der Fehlerbehandlung (bsp. ON ERROR...) am Programmanfang, springt der Interpreter bei Auftreten eines Fehlers zu der angegebenen Zeilennummer.

OR

Verwendungszweck:
Logischer Operator zur Oder-Verküpfung zweier Boolean-Ausdrücke, oder zum Vergleich zweier numerischer Bitmuster.

Syntax:
OR

Wirkungsweise:
Vergleicht gem. Wahrheitstabelle, ob eine - oder beide - Bedigungnen erfüllt sind.

Beispiele:

```
10 A = 1
20 B = 0
30 PRINT (A OR B)      Ergebnis: 1

10 A = 0
20 B = 1
30 PRINT (A OR B)      Ergebnis: 1

10 A = 0
20 B = 0
30 PRINT (A OR B)      Ergebnis: 0

10 A = 1
20 B = 1
30 PRINT (A OR B)      Ergebnis: 1
```

PI

Verwendungszweck:
Schlüsselwort als konstante Variable. Enthält die Zahl Pi.

Syntax:
(v) = PI

Wirkungsweise:
PI ist eine reservierte Konstante und enthält die Keiszahl Pi. Diese Zahl ist, wie die eulersche Zahl e (EE), fest im BASIC-Interpreter hinterlegt. Somit können Winkelfunktionen in Gleichungen schneller berechnet werden.

Beispiel:

```
X = PI
PRINT X   <CR>     Ausgabe:  3.1415926536
```

Weitere Besipiele:

```
PRINT PI          <CR>      Anzeige:  3.1415926536
PRINT PI/180                Anzeige:  .01745329252
PRINT 4*ATN(1)              Anzeige:  3.1415926536
```

POKE

Verwendungszweck:
Ein Byte direkt in den Rechner-Hauptspeicher (SRAM) schreiben.

Syntax:
POKE(adr),byte

Wirkungsweise:
Mit POKE(a),b wird auf einer gültigen 16-bit Memory-Adresse ein Byte (0...255d) in den Speicher geschrieben. Mit (b)=PEEK(a) kann das Byte wieder zurück gelesen werden. Die Variable „a“ steht hier für eine 16-bit Adresse (absolute Adressierung), mit Variable „b“ wird das Byte an die Speicherzelle im RAM übergeben. Beide Übergabeparameter sind Ganzzahlenausdrücke.

Write-Beispiel:

```
10 ADR = (&10000)
20 B = 123
30 POKE (ADR),B
```

Read-Beispiel:

```
10 ADR = (&10000)
20 B = PEEK (ADR)
30 PRINT B
```

Hinweis:
Der Wertebereich für PEEK- und POKE-Adressen beträgt 0...65525 (dezimal) bzw. 0000h...FFFFh. Der physikalische Speicher erstreckt sich bei SEPIA auf 8000h...FFFFh. Es sind jedoch Teil-Bereiche von 8000h...8300h und FE00h...FFFFh für den Interpreter reserviert. Wenn eine POKE-Anweisung einen wichtigen (bereits genutzten) Speicherbereich im RAM überschreibt, kann es zu einem Systemabsturz kommen, da intern genutzte Systemvariablen, Flags o.a. Parameter überschrieben werden.

POS

Verwendungszweck:
Die Funktion POS gibt die augenblickliche Spaltenposition des Text-Cursors zurück.

Syntax:
POS(0)

Wirkungsweise:
Der Spalten-Rückgabewert wird in der aktuellen Zeile berechnet, in der ein Text an den Bildschirm mit PRINT übergeben wurde.

Beispiel:

Position der Zeichenausgabe nur mit ';' möglich z.B.:

PRINT "UUUUUUUUU"; : PRINT POS(#0) <CR> Ausgabe: 9

PEEK

Verwendungszweck:
Ein Byte direkt aus den Rechner-Hauptspeicher lesen.

Syntax:
(v) = PEEK(adr)

Wirkungsweise:
Mit PEEK(a) wird auf einer gültigen 16-bit Memory-Adresse ein Byte (0...255d) ausgelesen. Die Variable „a“ steht hier für eine 16-bit Adresse (absolute Adressierung), die Variable „b“ enthält das Byte aus der gelesenen Speicherzelle. Beide Übergabeparameter sind Ganzzahlenausdrücke.

Write-Beispiel:

```
10 ADR = (&10000)
20 B = 123
30 POKE (ADR),B
```

Read-Beispiel:

```
10 ADR = (&10000)
20 B = PEEK (ADR)
30 PRINT B
```

Hinweis 1:
Der Wertebereich für PEEK- und POKE-Adressen beträgt 0...65525 (dezimal) bzw. 0000h...FFFFh. Der physikalische Speicher erstreckt sich bei SEPIA auf 8000h...FFFFh. Es sind jedoch Teil-Bereiche von 8000h...8300h und FE00h...FFFFh für den Interpreter reserviert. Wenn eine POKE-Anweisung einen wichtigen (bereits genutzten) Speicherbereich im RAM überschreibt, kann es zu einem Systemabsturz kommen, da intern genutzte Systemvariablen, Flags o.a. Parameter überschrieben werden.

Hinweis 2:
Mit PEEK können sowohl RAM-, als auch EPROM-Speicherzellen gelesen weden. Das Lesen von Speicherzellen verursacht keine Veränderung am Speicher. Mit PEEK können alle Speicheradressen von 0 bis 65535 gelesen werden. Der Inhalt wird dezimal ausgegeben. Andere Adressen ergeben die Fehlermeldung „nicht erlaubt“. Der Inhalt einer Speicherstelle kann auch einer Variablen zugewiesen werden.

PRINT PEEK(4149) <CR> oder PRINT PEEK(&1035) <CR>

Es wird der dezimale Inhalt der angegebenen Speicherstelle angezeigt.

PRINT

Verwendungszweck:
Text auf den Bildschirm (oder den Drucker mit PRINT#2) ausgeben.

Syntax:
PRINT #Kanal “Text“;(v),“x$“...
Befehlsform: PRINT # Kanal @ Satznummer, Konstante oder Variable oder Zeichenkette

Wirkungsweise:
Der PRINT-Befehl sendet den Inhalt von Variablen oder Text-Stings auf den Bildschirm.

Beispiele:

PRINT 123; A; „Testsatz“ <CR>

Am Bildschirm werden die Zahl 123, der Wert der Variablen A und das Wort „Testsatz“ jeweils mit zwei Leerzeichen Abstand von einander ausgegeben. Kommas zwischen den Zeichen oder Zeichenketten ergeben einen Ausdruck jede 14. Position, solange die Zeichenketten kürzer als 14 Zeichen sind.

PRINT#2:PRINT#2,“kurzer“;“Testsatz“ <CR>

Vom Drucker wird nach einer Leerzeile der Satz „kurzer Testsatz“ ausgegeben. Je nach Drucker müssen nach diesem Direktbefehl noch weitere Befehle oder eine Leerzeile folgen, damit der Ausdruck stattfindet. (bsp. Formfeed)

? 123 <CR>

An Stelle des Befehlswortes PRINT kann auch die Kurzform, ein Fragezeichen eingegeben werden. In BASIC-Zeilen (Programmiermodus) wird das Fragezeichen jedoch nicht durch PRINT ersetzt, auch nicht beim Speichern oder Laden von Programmen.

Hinweis:
Die formatierte Textausgabe erfolgt mit PRINT USING. Weitere Verweise: siehe TAB, SPC.

PRECISION

Verwendungszweck:
Festlegung der Zahlendarstellung und Rechengenauigkeit.

Syntax:
PRECISION(x)
Befehlsform: PRECISION (Zeichenanzahl)

Wirkungsweise:
Ergebnisse von mathematischen Berechnungen (z.B. Dezimalbrüche) werden mit maximal 11 Zeichen (ohne Punkt) ausgegeben. Diese Anzahl kann verringert werden, wobei die letzte ausgegebene Stelle gerundet ist. Intern wird mit 11 Zahlen weiter gerechnet. Die Zahl 0 oder keine Angabe nach dem Befehl ergeben wieder 11 Zeichen Genauigkeit.

Beispiel:

PRECISION 5 <CR> oder PRECISION (5)

Es werden bei Rechenergebnissen Zahlen mit maximal 5 Zeichen ohne Punkt ausgegeben.

PRINT 10/7 <CR> ergibt die Zahl 1.4286

PRINT 100/7 <CR> ergibt 14.286.

Mit PRECISION 0 <CR> wird nach PRINT 10/7 die Zahl 1.4285714286 ausgegeben.

PRIVACY

Verwendungszweck:
Security-Funktion. Schutz durch beliebiges Passwort. Alle Befehle einkapseln.

Syntax:
PRIVACY
Befehlsform: PRIVACY „Zeichenkette“ oder Variable

Wirkungsweise:
Durch die Angabe eines Schlüsselwortes (Password) an erster Stelle in irgend einer Programmzeile sind alle Befehle geschützt, mit denen der Inhalt eines BASIC-Programmes zugänglich gemacht oder verändert werden kann (AUTO, COPY, DELETE, EDIT, FIND, LIST, RENUMBER, REPLACE...). In allen Fällen wird die Fehlermeldung „nicht verfügbar“ ausgegeben. Auch das Löschen einer Zeile durch Eingeben der Zeilennummer mit nachfolgendem <CR> geht nicht. Das Schlüssel- bzw. Passwort kann irgendeine Konstante, Variable oder beliebige Zeichenkette sein.

Beispiel:

10020 Privacy „SeCrEt“ oder 10020 Privacy A$

In der Zeile 10020 wird als Schlüsselwort die Zeichenkette „SeCrEt“ definiert. Allen oben aufgeführten Befehlen muß unmittelbar nach dem Befehl das Schlüsselwort in Anführungszeichen folgen, danach können weitere, durch ein Komma getrennte Befehle angegeben werden, z. B. LIST „SeCrEt“,100-200.

READ

Verwendungszweck:
Datenfeld aus einer DATA-Anweisung lesen.

Syntax:
READ
Befehlsform: READ nummerische Variable, Stringvariable, durch Komma getrennt

Wirkungsweise:
Mit READ werden Daten aus einer zuvor getroffenen DATA-Anweisung an Variablen zugewiesen. Es können nummerische Ausdrücke, oder auch Zeichenketten übergeben werden. Dabei müssen jedoch die verwendeten Daten-Typen aus DATA und READ zwingend übereinstimmen.

Beispiel:

```
10 DATA 1,2,“ HUBER“,MEIER,,,4,ZZ
20 READ A,B,A$,B$,C,C$,D(1,1,1),D$(1,3)
30 PRINT A;B;A$;B$;C;C$;D(1,1,1);D4(1,3)
```

Anzeige: 1 2 HUBERMEIER 0 4 ZZ

Die Variable C enthält den Wert 0, C$ ist leer.

Hinweis:
Mit jeder READ-Anweisung wir der DATA-Zeiger automatisch um ein Element inkrementiert. Mit dem RESTORE-Befehl lässt sich dieser Zeiger wieder auf die erste Stelle zurückstellen. Nach der Zuweisung von Daten in einem DATA-Feld muss die READ-Anweisung nicht unmittelbar folgen. Wegen der besseren Übersicht sollte READ dennoch auf DATA folgen.

RESTORE

Verwendungszweck:
DATA-Zeiger einer DATA-Anweisung wieder auf ersten Wert rücksetzen.

Syntax:
RESTORE XYZ
Befehlsform: RESTORE Zeilennummer

Wirkungsweise:
Mit jedem Lesevorgang wird ein Zeiger auf die nächste Konstante innerhalb der DATA-Zeilen gesetzt. Am Ende der letzten DATA-Zeile eines Programmes zeigt der DATA-Zeiger ins „Leere“, ein weiterer Lesezugriff ergibt die Fehlermeldung „keine Daten Zeile XYZ“.

Ein RESTORE-Befehl setzt den DATA-Zeiger auf die erste Konstante zurück. Die Angabe der Zeilennummer einer DATA-Zeile setzt den Zeiger auf die erste Konstante in dieser Zeile zurück. Die Fehlermeldung „Sprungziel fehlt Zeile XYZ“ wird durch Angabe einer nicht vorhandenen BASIC-Zeile nach dem Befehl RESTORE verursacht.

Beispiel:

```
10 DATA 10,20,30
20 DATA 50,60,70
30 READ A,B
40 PRINT A,B
50 :
60 RESTORE 20    : REM hier wird das Datenfeld in Zeile 20 wieder rückgesetzt
70 :
80 READ A,B
90 PRINT A,B
```

RETURN

Verwendungszweck:
Rücksprung-Befehl einer GOSUB-Anweisung.

Syntax:
RETURN
Befehlsform: RETURN Zeilennummer

Wirkungsweise:
Der Befehl beendet die Sub-Routine einer GOSUB-Anweisung und keht eine Zeile unter dem GOSUB-Aufruf zurück, um das Programm an dieser Stelle weiter fortzuführen.

Beispiel 1:

```
100 A = 1
110 GOSUB 1000
120 PRINT A
130 END
140 :
1000 A = A +1
1100 RETURN
RUN
 2
```

Beispiel 2:

```
100 A = 1
110 GOSUB 1000
120 PRINT A
130 END
140 PRINT „Nix da!“
150 END
160 :
1000 A = A +1
1100 RETURN  140
RUN
 Nix da!
```

In Beispiel 2 wird die Zeile 120 nie erreicht. Stattdessen wird mit RETURN 140 das Programm dort weiter geführt. So kann über GOSUB-Ebenen hinweg gesprungen werden. Ebenso können auch noch mehrere Sprungziele mit angegeben werden, die in ihrer Reihenfolge nach und nach (Zeilenweise) abgearbeitet werden sollen.

Beispiel: 1100 RETURN 200,300,400,500...

RANDOMIZE

Verwendungszweck:
Der Befehl RANDOMIZE ergibt einen zufälligen Startpunkt der Pseudo-Zufallszahlenfolge aus dem CPU Refresh-Register.

Syntax:
RANDOMIZE

Wirkungsweise:
Übergabe des R-Registers als Startzahl für RND-Anweisung.

Beispiel 1:

```
10 RANDOMIZE
20 X = RND(1)
30 PRINT X;
40 GOTO 10

RUN
```

Es folgen zufällige Zahlenreihen zwischen 0 und 1.

Beispiel 2:

Der Bereich ganzzahliger Zufallszahlen kann durch Angabe eines Anfangswertes AW und eines größeren Endwertes EW begrenzt werden: B = INT((EW+1-AW)*RND+AW)

```
10 A = RND(1)
20 B = INT(A * 10)
30 PRINT B;
40 GOTO 10

RUN
```

Es folgen zufällige Ganzzahlenreihen zwischen 0 und 10.

REM

Verwendungszweck:
Kommentar Text zeilenweise ergänzen.

Syntax:
REM

Wirkungsweise:
REM-Kommentar wird vom BASIC-Interpreter während der Programmausführung vollständig ignoriert, verlangsamt aber geringfügig die Rechengeschwindigkeit während der Laufzeit. Leider kostet jeder Text Byte-für-Byte wertvollen Speicherplatz. Daher sollten Kommentare möglichst kurz ausfallen, wenn man nicht ganz darauf verzichten kann.

Beispiel:

```
10 REM Das ist nur ein Kommentar
20 REM der Zeile für Zeile eingegeben werden kann
30 REM
40 REM Kommentare können auch hinter einem Befehl plaziert werden
50 :
60 A = 123  : REM Hier wird der Variable A jetzt der Wert 123 zugewiesen
```

RESUME

Verwendungszweck:
Diese Anweisung unterstützt die Fehlerbehandlung und setzt das Programm nach einer Verzweigung in der Fehlerbehandlungsroutine weiter fort. Es wird der Befehl wiederholt, der den Fehler verursachte.

Syntax:
RESUME
RESUME 0
RESUME Zeilennunmmer
RESUME Next

Wirkungsweise:
Soll nach der Fehlerbehandlung in einem Unterprogramm mit der normalen Programmausführung fortgefahren werden, so muss der Befehl RESUME das Unterprogramm erst abschließen. Dieser Befehl entspricht dem RETURN nach einem GOSUB.

Hinweis:
Das Programm wird nach dem Befehl fortgesetzt, der den Fehler verursachte.
Am Anfang eines Unterprogrammes zur Fehlerbehandlung kann auch ein weiterer ON ERROR GOTO-Befehl stehen.

RENUMBER

Verwendungszweck:
Sortierbefehl, welcher die BASIC-Zeilennummern neu nummeriert.

Syntax:
RENUMBER
RENUMBER 100
RENUMBER 100,20

Wirkungsweise:
Werden während der BASIC-Programmierung weitere Zeilen in den Quelltext ergänzt, können schnell Platzprobleme entstehen. Mit RENUMBER erhält man weitere (freie) Zeilen. Die alten Zeilennummern werden dabei durch neue Nummer ersetzt.

Beispiel:

```
10 REM TEST
11 A = 44
12 B = 33
13 C = 77

RENUMBER
LIST

10 REM TEST
20 A = 44
30 B = 33
40 C = 77
```

Hinweis:
Mit RENUMER 100,20 wird das Listing dann ab Zeile 100 um jeweils 20 weiter inkrementriert.

```
100 REM TEST
120 A = 44
140 B = 33
160 C = 77
```

RUN

Verwendungszweck:
Start-Befehl für die BASIC-Anwendung.

Syntax:
RUN

Wirkungsweise:
Initialisiert alle Variablen, sowie Zeichenketten und startet das im RAM-Speicher befindliche BASIC-Programm ab der ersten Programmzeile.

Beispiel:

RUN

Hinweis:
Andere RUN-Befehlsformen werden nicht unterstützt. Alternativ kann man auch mit GOTO [Zeilennummer] starten. Dabei bleiben die bereits initialisierten Werte beibehalten.

RND

Verwendungszweck:
Einfacher Zufallszahlengenerator.

Syntax:
(x) = RND(1)

Wirkungsweise:
Ohne Argument liefert der RND-Befehl Pseudo-Zufallszahlen zwischen 0 und 1, die einem festgelegten Algorithmus entstammen. Dadurch ergeben sich nach dem Start des BASIC-Interpreters immer dieselben Zahlen.

Beispiel:

```
PRINT RND <CR>            Anzeige        :   .26943583396
  „     „     „               „            :   .72796806886
  „     „     „               „            :   .089675555242
```

Der Befehl RANDOMIZE ergibt einen zufälligen Startpunkt der Pseudo-Zufallszahlenfolge (Z80: Zahl aus dem Refresh-Register). Wird als Argument der Wert Null angegeben, wird erstmalig eine beliebige Zahl angezeigt und dann wiederholt.

```
PRINT RND(0) <CR>         Anzeige z.B.    : .5019531548
PRINT RND(0) <CR>             „           : .5019531548
```

Werte größer Null entsprechen dem RND-Befehl ohne Argument. PRINT RND(1) ergibt dieselben Zahlen wie PRINT RND. Negative Argumente bestimmen den Startpunkt der Zufallszahlenfolge. Die folgenden Zufallszahlensequenzen sind nach jedem BASIC-Start wiederum identisch.

```
A = RND(-280)  <CR> PRINT RND <CR> Anzeige    : 1.0207557995E-03
                    PRINT RND <CR>   „        : .94715060253
                    PRINT RND <CR>   „        : .826904723
```

Der Bereich ganzzahliger Zufallszahlen kann durch Angabe eines Anfangswertes AW und eines größeren Endwertes EW begrenzt werden: B = INT((EW+1-AW)*RND+AW)

RND1

Verwendungszweck:
Wie RND.

Syntax:
(x) = RND1(0)

Wirkungsweise:
Es werden die nachfolgenden Zahlen (z.B.: für Parameterübergaben) noch mit angehängt.

Beispiel:

? RND1(0)

Anzeige: .72796806886 1 0

RIGHT$

Verwendungszweck:
Für diesen Befehl gelten die gleichen Bedingungen wie für LEFT$(, nur wird jetzt der rechte Teil der Zeichenkette übernommen.

Syntax:
RIGHT$(„Zeichenkette“ oder Stringvariable, Zeichenanzahl 0 - 255)

Wirkungsweise:
Kopiert einen Teil-String in einen neuen String mit n Zeichen.

Beispiel:

```
10 A$ =“ABC123“
20 B$ = RIGHT$(A$,4)
30 PRINT B$

RUN
  C123
```

Hinweis:
Siehe auch MID$ und LEFT$.

REPLACE

Verwendungszweck:
Ersetzen von Zeichen im Programmteil.

Syntax:
REPLACE x$ y$ Nummer

Befehlsform:
REPLACE „alte Zeichenkette“ „neue Zeichenkette“, Zeilennummer oder Zeilenbereich

Wirkungsweise:
Eine bestimmte Zeichengruppe kann durch eine andere ersetzt werden. Dies erspart viele EDIT-Aufrufe, wenn z.B. in allen Programmzeilen einen Variablennamen durch einen anderen ersetzt werden soll. Die veränderten Zeilen werden angezeigt. Die Trennzeichen können verschiedenartig sein, sie sind bei dem Befehl FIND näher erläutert. Allerdings müssen Anfangs- und Endzeichen jeweils gleich sein. Da BASIC-Befehle nur in Großbuchstaben in Programmen vorhanden sind, werden klein-geschriebene Zeichenketten natürlich nicht gefunden. Die maximale Zeichenanzahl von 254 Zeichen darf durch die Veränderung nicht überschritten werden, sonst ergibt dies die Fehlermeldung „String zu lang“.

Beispiel:

REPLACE ‘A$’“B$“,10-2300

Hinweis:
Die Variable A$ wird, soweit vorhanden, in den Zeilen 10...2300 durch die Variable B$ ersetzt. Sind Hochkommas in einer Zeichenkette zu ersetzen, müssen andere Trennzeichen wie z.B. (‘) verwendet werden.

STOP

Verwendungszweck:
Laufendes Programm an einer vorbestimmten Stelle (Zeile) stoppen.

Syntax:
STOP

Wirkungsweise:
STOP beendet das Programm in der Zeile, in der der Befehl vom Interpreter ausgeführt wird. Alle Variablen und Stringinhalte bleiben erhalten, solange keine Veränderung am Quellcode durchgeführt wird.

Beispiel:

```
10 A = 12345.67890
20 STOP

RUN

* Abbruch *  Zeile 20

? A    <CR>      Anzeige: 12345.6789
```

SIN

Verwendungszweck:
Die SIN(x) Funktion liefert den mathematisch Sinus ihres nummerisch Arguments. Das Argument muss in Bogenmass angegeben sein. Bei Altgrad verwendet man einen Umrechnungsfaktor mit (PI/180).

Syntax:
n = SIN(x)

Wirkungsweise:
Das als Argument angegebene Bogenmaß wird in den Sinus umgerechnet.

Beispiel 1:

PRINT SIN(45*PI/180) <CR> Anzeige: .70710678119

Beispiel 2:

```
10 INPUT „Bogenmass in Grad: “ ; X
20 A = SIN(X * 0.0174533)
30 PRINT „Grad > nach > Altgrad = “ ; A

RUN
```

SGN

Verwendungszweck:
Mathematisches Vorzeichen eines nummerischen Arguments bestimmen.

Syntax:
SGN(n)

Wirkungsweise:
Die Signum-Funktion (SGN) ergibt, unabhängig des Vorzeichens, folgende Zahlen für numerische Argumente aus:

bei positiven Zahlen = 1
bei negativen Zahlen = -1
bei Null = 0

Beispiele:

PRINT SGN(123) <CR>
Anzeige: 1

PRINT SGN(-4.56) <CR>
Anzeige: -1

PRINT SGN(0) <CR>
Anzeige: 0

Hinweise:
Wird keine Zahl in Klammern angegeben, so erfolgt Fehlermeldung „Syntaxfehler“
Wird keine Klammer angegeben, so erfolgt die Rückmeldung mit NULL.
Es werden nur Zahlen bis zu einem Wert von -1e+38 bis 1e+38 vearbeitet.

SQR

Verwendungszweck:
Quadratwurzel aus n berechnen.

Syntax:
SQR(n)

Wirkungsweise:
Es wird die Quadratwurzel der als Argument angegebenen Zahl (n) berechnet. Negative Zahlen ergeben die Fehlermeldung „nicht erlaubt“. Höhere Wurzeln lassen sich durch den inversen Exponenten der Zahl X berechnen.

Beispiele:

PRINT SQR(2)	<CR>	Anzeige: 1.4142135624
PRINT SQR(5)*SQR(5)	<CR>	Anzeige: 5
PRINT 10000^(1/4)	<CR>	Anzeige: 10 (= 4. WURZEL)

STR$

Verwendungszweck:
Die im Argument X enthaltenen Zahlen werden mit diesem String-Befehl als Zeichenkette gekennzeichnet.

Syntax:
STR$(x)
STR$ (Zahl oder nummerische Variable oder Ausdruck)

Wirkungsweise:
Wandelt Zahl in einen Text-String um. Somit verhält sich STR$(x) komplementär zu VAL($).

Beispiel 1:

```
A$ = STR$(123456)          <CR>
PRINT A$                   <CR>        Anzeige: 123456
PRINT ASC(LEFT$(A$,1) <CR>             Anzeige: 32
```

Das erste Zeichen der Zeichenkette A$ ist ein Leerzeichen, was oft zu einem Fehlverhalten eines BASIC-Programmes führt.

Beispiel 2:

```
10  INPUT  „Eine  Zahl eingeben“ ; Z
20  ON LEN ( STR$ ( Z ) )  GOSUB 100 , 200 , 300
30 :
100 PRINT „100...“
110 RETURN
200 PRINT „200...“
210 RETURN
300 PRINT „300...“
310 RETURN
```

Dieses kleine Besipielprogramm verzweigt mit GOSUB zu verschiedenen Unterprogrammen, abhängig von der Anzahl der Zeichen, die als Variable Z eingegeben wurden.

STRING$

Verwendungszweck:
Textausgabe eines ASCII-Zeichen als Zeichenkette.

Syntax:
STRING$ („Zeichenkette“ oder Stringvariable , Anzahl X)

Wirkungsweise:
Eine bestimmte Zeichenkette wird mit der angegebenen Anzahl X „multipliziert“, d.h. die neue Zeichenkette enthält X-mal die alte Zeichenkette. STRING$ eignet sich ebenso zur formatierten Ausgabe eines oberen und/oder unteren Randes auf dem Bildschirm bzw. auf dem Drucker.

Beispiel:

```
10 A$ = „*-“
20 B$ = STRING$(A$,10)
30 PRINT B$
```

Anzeige: *-*-*-*-*-*-*-*-*-*-

Die Summe aller Zeichen darf die Zahl 255 nicht überschreiten.
Sonst erfolgt die Fehlermeldung: „nicht erlaubt Zeile 20“

Hinweis:
Es sind nur ganzzahlige Ausdrücke im Bereich von 0 bis 255 erlaubt.

STEP

Verwendungszweck:
Schrittweite in FOR...NEXT Schleifen festlegen.

Syntax:
STEP(n)

Wirkungsweise:
FOR-NEXT-Befehle dienen zum Aufbau von Befehlsschleifen. Mit STEP wird die Schrittweite des Zählers bestimmt, mit der die Schleife inkrementriert wird.

Beispiel:

Befehlsform:

FOR	Laufvariable = Anfangswert
TO	Endwert
STEP	Schrittweite
...	beliebige Befehle
NEXT	Laufvariable

```
10 FOR I = -1 TO 1 STEP 0.1
20 PRINT I,“TEST“
30 NEXT I
```

Die Angabe der Laufvariable nach dem Befehl NEXT kann bei nur einer Schleife entfallen. Bei verschachtelten Schleifen muß die jeweilige Laufvariable angegeben werden. Solche Schleifen müssen nach dem Schema FOR A..FOR B..FOR C..usw - NEXT C..NEXT B.. NEXT A aufgebaut sein. Da die Rücksprungadressen der Schleifen in einem bestimmten Speicherbereich (Stack) zwischengespeichert werden, ist auch die Anzahl der verschachtelten Schleifen begrenzt.

```
 5 DIM A(15,15,15)
10 FOR I = 1 TO 15
20 FOR K = 1 TO 15
30 FOR L = 1 TO 15
40 A(I,K,L) = 1
50 NEXT L,K,I
```

Eine dreidimensionale Matrix wird mit dem Wert 1 aufgefüllt. Mehrere Laufvariablen können, durch ein Komma getrennt, in eine Zeile geschrieben werden.

SPC

Verwendungszweck:
Zwischenräume mit Leerzeichen (Space) auffüllen.

Syntax:
SPC(n)
SPC(: Befehlsform: SPC(Zahl der Leerzeichen)

Wirkungsweise:
Zur Ausgabe von Zwischenräumen dient der Befehl SPC.

Beispiel:

PRINT „TEST“;SPC(10);“TEST“ <CR>

Anzeige: TEST TEST

In diesem Beispiel werden 10 Leerzeichen zwischen beiden TEST-Texten eingefügt.
Dies spart Speicherplatz, da kein weiterer String mit 10 Leerzeichen angelegt werden muss.

Hinweis:
Eine Zuweisung zu einer Variablen ergibt die Fehlermeldung „Syntaxfehler Zeile XYZ“.
Es dürfen max. 255 Leerzeichen für SPC(n) als Ganzzahl definiert werden, also n < 256.
Ich n > 255, wird die Fehlermeldung „nicht erlaubt Zeile XYZ“ ausgegeben.

TAN

Verwendungszweck:
TAN liefert den Tangens von *x* im Bogenmaß.

Syntax:
TAN(*x*)

Wirkungsweise:
Für den im Bogenmaß angegebenen Wert (x) wird der Tangens berechnet.

Beispiel:
PRINT TAN(45*PI/180) <CR> Anzeige: 1

Hinweis:
Bei einem Überlauf von TAN wird eine entsprechende Fehlermeldung ausgegeben und die größte, darstellbare Zahl mit entsprechendem Vorzeichen zurückgegeben. Das Programm wird dennoch fortgesetzt. Wenn *x* als Gradangabe vorliegt, erfolgt die interne Berechnung von TAN(*x*) mit TAN(*x**pi/180).

TRACE

Verwendungszweck:
Ausgabe (mitverfolgen) der bearbeiteten Zeilennummern während der Laufzeit des Interpreters.

Mit TRACE können schwerwiegende Programmierfehler während der Bearbeitung untersucht werden, um beispielsweise Programmabstürze noch vor dem Absturz zu sichten.

Syntax:
TRACE
Befehlsform: TRACE # Kanal @ Satznummer , Wert

Wirkungsweise:
Die BASIC-Zeilennummern der vom Interpreter bearbeiteten Programmzeilen werden nach dem RUN-Befehl nacheinander auf den angegebenen Kanal ausgegeben, wenn für den Parameter WERT die Zahl 1 eingesetzt wurde.

TRACE wird durch den WERT 0 wieder abgeschaltet.

Beispiel:

```
10 REM Text
20 PRINT „Schalte TRACE in der naechsten Zeile an...“
30 TRACE 1
40 REM ....
50 PRINT „Schalte TRACE in der naechsten Zeile wieder ab...“
60 TRACE 0
70 REM Fertig
80 STOP
```

Hinweise:

TRACE 1 <CR>
Ausgabe der Zeilennummern der bearbeiteten Programmzeilen in Spitzklammern am Bildschirm (<10><20><30><145> usw.). Trace 0 schaltet den Befehl wieder ab.

TRACE#2,1 <CR>
Ausgabe der Zeilennummern auf den Drucker (LPT1).

TRACE verhält sich ähnlich wie TRON / TROFF bei anderen BASIC-Varianten.

TO

Verwendungszweck:
Siehe FOR - Schleifenbefehl.

FOR ... TO ... STEP ... EXIT ... NEXT
Diese Befehle dienen zum Aufbau von Befehlsschleifen.

Befehlsform:

FOR	Laufvariable = Anfangswert
TO	Endwert
STEP	Schrittweite
...	beliebige Befehle
NEXT	Laufvariable

THEN

Verwendungszweck:
IF-THEN-Anweisung. Logische Weiterverzweigung bei Wahr (True).

Syntax:
IF (Wahr) THEN GOTO (Verzweigung) ....

Wirkungsweise:
IF weist das Programm an, den nachfolgenden logischen oder Vergleichsausdruck zu prüfen. Ergibt die Auswertung 1 (= Wahr bzw. True) , so führt der BASIC-Interpreter die nachfolgende Anweisung entsprechend aus - andernfalls wird die Anweisung zur Verzweigung ignoriert bzw. übersprungen. THEN verknüpft bei „1“ (Wahr) diese Anweisung mit einem weiteren BASIC-Schlüsselwort: z.B. GOTO oder GOSUB. Es können jedoch auch andere Anweisungen oder Ausgaben stattfinden, oder ein Numerischer Ausdruck in einer Variablen verändert werden.

Beispiel 1:
```
10 A = 10
20 B = 20
30 IF (A + B) = 30 THEN GOTO 100
40 : REM ...
100 PRINT „Das Ergebnis ist 30“
```

Beispiel 2:
```
10 A = 10
20 B = 20
30 C = 0
40 IF (A + B) = 30 THEN C = (A + B)
50 GOTO 100
100 PRINT „Das Ergebnis ist “; C
```

Beispiel 3:
```
10 A = 10
20 B = 50
30 FOR X = A TO B
40 IF X = 30 THEN PRINT „Das Ergebnis ist „;X
50 NEXT
60 END
```

TAB

Verwendungszweck:
Tabulator-Befehl. Cursor springt auf die (Tabulator-) Position *n.*

Syntax:
TAB(*n*)
TAB(: Befehlsform: TAB(Position X ab linkem Rand);

Wirkungsweise:
Bei der Ausgabe eines Zeichens am Bildschirm (oder auf den Drucker) wird der Cursor (oder Druckkopf) immer vom linken Rand aus X Zeichen nach rechts versetzt bzw. neu positioniert.

Beispiel:

Im Beispiel kann der zweite Befehl TAB(5) nicht ausgeführt werden.

PRINT TAB(10);“#“;TAB(5);“+“ <CR> Anzeige: _____#+

10 Leerzeichen

Hinweis:
Liegt die gültige Ausgangsposition bereits hinter der Position *n*, geht TAB zur nächsten Position *n* in der nächsten Zeile vor.

1 ist die am weitesten links liegende Position. Die am weitesten rechts liegende Position wird von der Bildschirm- oder Druckerbreite vorbestimmt.

n muß im gültigen Bereich zwischen 1 bis 255 liegen.

Steht die TAB Funktion am Ende einer Liste von Datenfeldern, setzt der Cursor die Schreibmarke nicht in die nächste Zeile. TAB kann nur mit den PRINT-Anweisungen verwendet werden.

USING

Verwendungszweck:
USING erlaubt eine umfangreiche Formatierung von Zahlen und Texten in Tabellenform.

Syntax:
PRINT # Kanal, USING Zeilennummer XYZ; Variable 1, Variable 2, usw...
Befehlsform: XYZ ! Formatierungsangabe (#,.,+,-,**,$$,**$,',^^^^,L,R,C,E)

Wirkungsweise:
Bei der Ausgabe von Zahlen und Zeichenketten ergibt sich die Anordnung aus der Zeichenanzahl. Ein Tabellenaufbau ist daher oft nur mühsam mit entsprechenden String-operationen möglich. Der Befehl USING bietet eine einfache Tabellierungsanweisungen, die sogar im Programm noch gewechselt werden können.

Beispiel:
A$ = „Hallo " : A = 12345.67890 <CR>
PRINT USING A$; "#####.#",A <CR> Anzeige: Hallo 12345.6

Hinweise:

#	Platzhalter für eine anzuzeigende Ziffer. Hat die darzustellende Zahl weniger Ziffern, wird die Zahl rechtsbündig gestellt und links mit Leerzeichen aufgefüllt. Paßt eine Zahl nicht in das Nummernfeld, wird sie doch ganz ausgegeben und durch ein vor die Zahl gestelltes Prozentzeichen (%) gekennzeichnet.
.	Punkt. Setzt die Position des Dezimaltrennzeichens (nur Zahlenausgabe).
+	Reserviert Platz für ein Vorzeichen zu einer Zahl. Das Vorzeichen wird dann immer mit ausgegeben.
-	Wird Minus hinter einen Ausdruck verwendet, wird das Vorzeichen hinter statt vor der Zahl positioniert.
**	Zwei Sternchen am Anfang eines Nummernfeldes bedeuten, dass an jede nicht besetzte Position des Nummernfeldes ein Asteriks-Sternchen (*) gesetzt wird (z.B. für Zahlkarten üblich).
$$	Vor einer Zahl plaziert, bewirkt, dass ein Leerzeichen mit Dollarzeichen vor der Zahl ausgegeben wird.
**$	Kombiniert die Formatierung ** und $.
,.	Ein Komma links vom Dezimalpunkt im Nummernfeld bedingt, dass alle 3 Zeichen ein Komma gesetzt wird.
^^^^	Vier Pfeile am Ende der Kennzeichnung des Nummernfeldes deuten an, dass die Zahl immer in exponentialer Schreibweise angegeben wird (E+nn oder E-nn). Ein Dezimalpunkt ist erlaubt.
L	Textformatierungsanweisung: linksbündig. Befehlsform: 'LLLL Die Anzahl der L-Buchstaben ergibt die max. Länge der Zeichenkette, eine längere wird rechts abgeschnitten.
R	Textformatierungsanweisung: rechtsbündig. Befehlsform: 'RRRRRRRR Die Anzahl der R-Buchstaben ergibt die max. Länge der Zeichenkette, eine längere wird links abgeschnitten.
C	Textformatierungsanweisung: zentrieren. Befehlsform: 'CCCCCCCCCCCCCCCCCCCC Die Zeichenkette wird im C-Feld zentriert, eine zulange Zeichenkette wird rechts abgeschnitten.
E	Textformatierungsanweisung: linksbündung + ergänzen. Befehlsform: 'E Es braucht nur ein E angegeben werden, da die gesamte Zeichenkette ausgegeben wird.

VAL

Verwendungszweck:
VAL Liefert den numerischen Wert einer Zeichenfolge (Stringvariable), oder Zeichenkette.

Syntax:
VAL(A$)

Wirkungsweise:
VAL(Str$) dient zur Umwandlung einer String-Zahl in einen nummerischen Wert. Dabei entfernt die VAL-Funktion automatisch führende alle Leerzeichen, Tabulatoren und Zeilenvorschübe aus der String-Zeichenfolge. VAL(Str$) ist quasi eine Komplementärfunktion von STR$.

Beispiel:

Die Fehlermeldung „Datentypen (Zahl/String) vermischt“ erscheint, wenn man einer nummerischen Variablen den Inhalt einer Zeichenkette zuordnen will. Mit dem Befehl VAL ist dies möglich. Es werden jedoch nur Zahlen am Anfang einer Zeichenkette ausgewertet.

```
A$ = „45.7“       <CR>
PRINT VAL(A$)             <CR>   Anzeige: 45.7

PRINT VAL(„1234ABC“)      <CR>   Anzeige: 1234

PRINT VAL(„ABC1234“)      <CR>   Anzeige: 0

PRINT VAL(„123.45A“)      <CR>   Anzeige: 123.45
```

Hinweis:
Handelt es sich beim ersten Zeichen nicht um einen numerischen Wert, liefert VAL=0.

VARPTR

Verwendungszweck:
VARPTR ist eine Adressenangabe von Variablen und Stringzeigern.

Syntax:
VARPTR (Variable)

Wirkungsweise:
Die dezimale Adresse der Inhalte von Variablen und Matritzenelementen im Speicher wird mit diesem Befehl ermittelt und kann z.B. direkt bei den Befehlen PEEK und POKE angewendet werden. Bei Variablen für Zeichenketten (z.B: A$) wird die Adresse des Stringzeigers ermittelt und ausgegeben. Das erste Byte an dieser Adresse enthält die Länge der Zeichenkette, das nächst folgende Byte ist immer 0. Die dann folgenden beiden Bytes enthalten die Adresse der Zeichenkette, das LOW-Byte zuerst. Die letzten beiden Bytes werden nicht benützt.

Beispiel:

```
10 A = 2
20 A$ = „TEST“

PRINT VARPTR(A) <CR>
Ausgabe der dezimalen Adresse

PRINT VARPTR(A$) <CR>
Ausgabe der dezimalen Adresse des Stringzeigers
```

Hinweis:
VARPTR(*Variable*) liefert hier die Zeiger-Adresse des ersten Datenbytes der angegebenen Datenvariable. Vor der Ausführung der VARPTR-Funktion muß dem Variablennamen ein zulässiger Wert zugewiesen werden, anderenfalls wird eine Fehlermeldung ausgegeben.
Der Variablenname kann von beliebigem Typ sein. Bei der zurückgegebenen Adresse handelt es sich immer um einen ganzzahligen Wert zwischen 0 und 65535. Vor dem Aufruf sollten alle einfachen Variablen bereits zugewiesen sein, da sich die Adressen jedesmal ändern, wenn wieder neue Variablen zugewiesen werden.

WAIT

Verwendungszweck:
Schleifenbefehl unterbricht die Programmausführung und wartet bis der Vergleichswert mit dem Eingangsport (I/O-Port) übereinstimmt.

Syntax:
WAIT (Adresse A, Maske M, Zustand Z)

Wirkungsweise:
Der Interpreter wartet in einer Schleife, bis ein Eingangsport mit der dezimalen Adresse A in den mit der Maske M angegebenen Bits den Zustand Z erreicht hat. Erst dann wird die nächste Programmzeile weiter bearbeitet. So können z.B. direkt vom Port A der PIO einzelne Bits untersucht werden, ob sie den Zustand 0 oder 1 besitzen. Der vom Port A eingelesene Wert wird mit dem Wert Z EXKLUSIV-ODER verknüpft und das Ergebnis mit dem in M angegeben Wert UND verknüpft. Solange das Ergebnis gleich 0 ist, wird das Ganze wiederholt. Erst dann geht es zum nächsten Befehlsabschnitt. Wird der Parameter Z nicht angegeben, so ist Z=0. Die Werte A, M und Z müssen dezimal angegeben werden (oder sedezimal mit „&“ angeben).

Beispiele:

```
100 WAIT  32 , 2

RUN
```

Das Programm wird solange unterbrochen, bis Hardware-Port 32 eine 2 als Eingabe erhält.

Hinweis:
Als INPUT-Adresse ist nur eine gültige Anschlußnummer (Portnummer) des Computers im Bereich 0 bis 255 erlaubt. M und Z sind ganzzahlige Ausdrücke im Bereich 0 bis 255. Bei der Ausführung des WAIT-Befehls wird die Maske auf gesetzte Bits überprüft. Ist eines der Bits gesetzt, wird die Programmausführung mit der nächsten Anweisung fortgesetzt. WAIT wartet nicht, bis ein komplettes Bitmuster ansteht. Wenn nur ein Bit überprüft werden soll, genügt bereits der Vergleich mit diesem Bit, um den Programmablauf fortzuführen. Mit der WAIT-Anweisung besteht die Möglichkeit, eine Endlosschleife einzubauen. Zum manuellen Verlassen der WAIT-Schleife kann entweder CTRL-E eingegeben, oder aber das System per Hardware-RESET zurückgesetzt werden.

XOR

Verwendungszweck:
Operator. Logische Exklusiv-Oder-Verknüpfung.

Syntax:
(n1) XOR (n2)

Wirkungsweise:
XOR steht für exklusives OR. Wenn jedoch beide n-Vergleichswerte 1 sind, wird die Ausgabe rückgesetzt (Ergebnis=0).

Beispiel:

```
10 A = 1
20 B = 1
30 C = A XOR B
40 PRINT C

RUN

Anzeige: 0
```

So wie bei „OR“ setzt der XOR-Operator im Ergebnis das Bit, wenn eines der beiden Bits in N1 und N2 gesetzt ist. Sind jedoch beide N-Bits gesetzt, oder beide N-Bits = NULL, wird das Bit im Ergebnis nicht gesetzt.

Hinweis:
XOR wird bei Vergleichen angewendet, wenn eine Aussage erfüllt sein muss, jedoch nicht beide Bedingungen erfüllt sein dürfen.

X..... Befehle

siehe Advanced-BASIC in Kapitel 1.